THESIS

AUTONOMOUS AGENTS UTILIZING PASSIVE OPTICAL SENSING FOR CONTROL

by

Aaron M. Stalford

March 2019

Thesis Advisor:	Vladimir N. Dobrokhodov
Second Reader:	Brian S. Bingham

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE AUTONOMOUS AGENTS UTILIZING PASSIVE OPTICAL SENSING FOR CONTROL	**5. FUNDING NUMBERS**
6. AUTHOR(S) Aaron M. Stalford	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The project addresses the problem of integrating a passive optical sensor onboard an autonomous unmanned aerial vehicle (UAV) and its operation in future missions against various targets/threats. The thesis develops the analytical and experimental software and hardware framework around a commercial-off-the-shelf quadcopter (COTSQ) for autonomous missions. The COTSQ features advanced autopilot capabilities along with an onboard optical sensor that is fully integrated into the control software. This research is applicable to any agent that has similar passive optical sensing capability. The work addresses the following key research questions and concepts:

- Prove the sufficiency of the inner-outer loop design of a COTSQ autopilot to perform the autonomous sensing and detection of targets and threats in real time
- What is the desired architecture of a high-level guidance controller to enable a single COTSQ and then multiple COTSQs to optimally search and automatically detect targets/threats?
- How can the optical feed be used to facilitate detection and tracking of objects based on the real-time video feed from the UAVs?
- How can this optical feed be utilized to advance the intelligent autonomy of the UAVs?
- What are the fundamental limitations of achievable performance of the following components?
    - o Onboard instrumentation to enable robust execution of the search & detection mission and its robust scalability to potentially unlimited number of UAVs .

14. SUBJECT TERMS quadcopter, drone, control, search, search theory, optical sensing, video processing, passive sensing, multicopter, OpenCV, opencv, ROS	**15. NUMBER OF PAGES** 119
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AUTONOMOUS AGENTS UTILIZING PASSIVE OPTICAL SENSING FOR CONTROL

Aaron M. Stalford
Lieutenant, United States Navy
BS, University of South Florida, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2019

Approved by: Vladimir N. Dobrokhodov
Advisor

Brian S. Bingham
Second Reader

Garth V. Hobson
Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The project addresses the problem of integrating a passive optical sensor onboard an autonomous unmanned aerial vehicle (UAV) and its operation in future missions against various targets/threats. The thesis develops the analytical and experimental software and hardware framework around a commercial-off-the-shelf quadcopter (COTSQ) for autonomous missions. The COTSQ features advanced autopilot capabilities along with an onboard optical sensor that is fully integrated into the control software. This research is applicable to any agent that has similar passive optical sensing capability. The work addresses the following key research questions and concepts:

- Prove the sufficiency of the inner-outer loop design of a COTSQ autopilot to perform the autonomous sensing and detection of targets and threats in real time
- What is the desired architecture of a high-level guidance controller to enable a single COTSQ and then multiple COTSQs to optimally search and automatically detect targets/threats?
- How can the optical feed be used to facilitate detection and tracking of objects based on the real-time video feed from the UAVs?
- How can this optical feed be utilized to advance the intelligent autonomy of the UAVs?
- What are the fundamental limitations of achievable performance of the following components?
    - Onboard instrumentation to enable robust execution of the search & detection mission and its robust scalability to potentially unlimited number of UAVs.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

COTS	Commercial-off-the-Shelf
COTSQ	Commercial-off-the-Shelf Quadcopter
CVBridge	Computer Vision Bridge
DOF	Degrees of Freedom
GCS	Ground Control Station
LOS	Line of Sight
LRC	Lateral Range Curve
OpenCV	Open Source Computer Vision Library
PWM	Pulse Width Modulation
R&D	Research and Development
RGB	Red Green Blue
ROS	Robot Operation System
RPMs	Revolutions per Minute
TOI	Target of Interest
UAVs	Unmanned Aerial Vehicles
uint8	Unsigned Integer 8 Bit
VIO	Visual Inertial Odometry
VMCS	Vicon Motion Capture System

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife, Meghan Stalford, for all her assistance, support, love and understanding through the thesis process.

I would also like to thank my two daughters, Shannon and Keira, for sacrificing many weekends with me so that I could advance my education.

I would like to thank my thesis advisor, Associate Professor Vladimir Dobrokhodov of the Mechanical Engineering department of Naval Postgraduate School. Professor Dobrokhodov was willing to take me on when other resources failed. His door was always open and he was always willing to guide, teach or correct. Without his mentorship this thesis would not have been possible.

I would like to thank Professor Brian Bingham for his mentorship of my thesis and continual guidance that allowed me to improve my knowledge and writing ability.

I would like to thank Lee Van Houtte, who was instrumental in acquiring and maintaining all the physical components for this thesis.

I would also like to thank Patrik Pietschmann and Rousseau for the numerous songs they have composed on YouTube that allowed me to get through writing this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

The United States Navy has focused on autonomous platform use in the air, land and sea to conduct missions such as surveillance, detection, maintaining military presence, and search and rescue [1]. One major problem with the current state of technology is the existential cost of military autonomous platforms. The cost is represented by both the research and development (R&D) efforts and the cost of highly trained personnel required to operate the unmanned systems. This thesis addresses a specific dimension of the R&D cost that is defined by the capabilities of a single autonomous platform. Special attention is paid to the scalability of the UAV that can potentially provide exponentially higher system performance when the number of autonomous platforms used in cooperative missions is increased. The extension of an unmanned vehicle to cooperation or swarming allows not only higher expected system level performance but also enables graceful degradation of performance of the system when a single unmanned vehicle or a subset of the swarm is lost during the mission. Typically, mission planners and decision makers are not as willing to lose high-dollar assets. What if the platforms were smaller, much more affordable, and almost as capable as their larger counterparts? What if these capable drones are used cooperatively to achieve the mission objective? What are the bounds of their cumulative capability and how does the swarm performance degrade with the loss of its members?

The commercial-off-the-shelf quadcopter (COTSQ) has gained popularity in the private sector for everything from hobby flying to 4K video-making tasks. These COTSQs have advanced sensors typical of full-scale manned systems, such as GPS, accelerometers, gyroscopes, magnetometers, and miniature HD video cameras. They also come with advanced control features that allow them to hover in one place when the user is not providing input, follow a person while videoing, and transit from one waypoint to another via pre-programmed paths. These COTSQs have nearly achieved capabilities of more expensive military-grade drones. Can these COTSQs be changed or modified in a way that allows the U.S. military to control them autonomously for military operations? If so, having autonomous capability in a COTSQ would greatly reduce the costs from tens or hundreds of thousands of dollars for each autonomous platform to below a thousand dollars for each platform. This cost savings would allow for more field experimentation and the advancement of technology in this field as the decision makers would not be as afraid of losing a lower-dollar asset. The decreased fear of losing a high-value asset would also allow many more autonomous platforms to be deployed by the military members that will be using them every day. Advancements will be made much quicker due to larger numbers of realistic experiments and data that is received from each of those experiments.

B. THESIS OVERVIEW

This thesis developed control architecture, both hardware and software, that can be used across numerous platforms in air, land, and sea. The focus of this thesis is on aerial platforms; however, the theory can be extended to other platforms equipped with passive optical sensors. In the envisioned scenario the COTSQ was used as an ISR platform. The area of search was predetermined and was used as bounds for the experiment. A single quadcopter was deployed in the given search area utilizing a predetermined search path. The quadcopter then followed this predetermined search path and looked for Targets of Interest (TOI). The forward-looking camera was used as the detection sensor in this case in conjunction with the visual detection software. All other sensors were used by internal controllers to create stable flight conditions. In order to get to this point, several fundamental capabilities had to be completed along the way.

First, the COTSQ was taken from the traditional COTS configuration and changed to an autonomous configuration. This required performing experiments to understand the communication paths and the flight dynamics of the COTSQ. Experiments were also performed to understand the key requirements of integrating the front-facing high definition camera with the visual detection and real-time flight control software. This thesis gives the basics for understanding the operational limitations of the COTSQ in military missions. This information also provides a detailed understanding of the expected performance of the inner loop controller (autopilot) on the COTSQ.

The next challenge was to determine what path planning approach should be taken and to design a controller to execute that approach. The onboard autopilot was used in its native configuration. A ground control station (GCS) was developed that provides higher level commands to the quadcopter. The GCS autonomously operates the outer loop controller, path planning controller, sequencer of waypoints, and the visual detection software. These high-level control actions could be performed onboard but would require extensive changes to the proprietary software on the COTSQ. The embedded implementation is beyond the scope of this thesis.

Once the COTSQ was able to be flown in autonomous flight, attention was shifted to the search theory. The different search theories had to be analyzed for the most effective search path for the given geometry of the search area and the detector; then the TOI geometry, shape, and color had to be selected so that the detector (High Definition Camera) could see the shape independently from the background in conjunction with the visual detection software. Once all these pieces were in place, the final experiments were performed. The different pieces from the individual flight paths, the video tests, and integration of the two were combined in the culmination of the final search test.

C. LITERATURE REVIEW

The task of computer vision and its integration in control is certainly not a new research topic that is represented by hundreds of research publications and commercial products available on the market. However, the development of vision-based processing and its integration with safety-critical real-time flight control onboard of small UAVs with limited signal processing capabilities is a relatively new area represented by some growth of research published started in the last 8–10 years. In 2008, a group from Brigham Young, Stanford and Vanderbilt Universities used a fixed-wing UAV with video capabilities to perform search and rescue research [2]. In this experiment, the fixed-wing UAV flew high above a wilderness area performing a search in order to improve detection algorithms. In 2010, a group from the University of Oxford used quadcopter vision detection to map and search areas for human rescue operations [3]. In 2015, a group form National School Polytechnic in Algeria was able to use an AR Drone 2.0 to detect color in the quadcopter video feeds. This group was also able to use fiducial markers as detection targets for the quadcopter [4].

D. THESIS CONTRIBUTIONS

1. Hardware and Software Architecture

The hardware and software architecture that was developed to modify the COTSQ to a fully autonomous platform is detailed in Chapter III. The architecture integrates guidance (optimal search), navigation, and control (path following) software with essential vision-based detection and identification algorithms all running in real-time. The complete architecture allows for an agile video platform to be used for target search and detection. Successful demonstration of the experiments culminated in the effective integration of all components into an inexpensive passive optical search tool.

2. Optical Lateral Range Curve Analysis

The optical lateral range curve (LRC) testing and analysis is detailed in Chapter IV. The LRC test is a systematic approach of quantifying the lateral range or sweep width of an optical sensor. The LRC method provides a way of making a non-uniform sensor into a cookie cutter sensor that can be used for search and detection optimization.

3. Vector Path Following Algorithm

As a part of the overall "mission controller," the quadcopter had to be able to join and follow lines in order to execute the search and detection mission. The path following algorithm was expanded from a 2D approach to a 3D approach. This new 3D approach uses vector geometry to guide the quadcopter to the line rather than heading error and distance error to the line. This new path following algorithm is detailed in Chapter III.

4. Vision-Based Algorithms Onboard an Agile UAV

The vision detection architecture provides another entry point into the world of robotic vision. Having an HD video feed that is accessible for real-time analysis on a highly agile quadcopter increases the speed and range of robotic vision and detection. All of the analysis tools that are being used in robotic vision including machine learning, face and object recognition, and edge detection can be used on this agile platform.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BASIC QUADCOPTER COMPONENTS

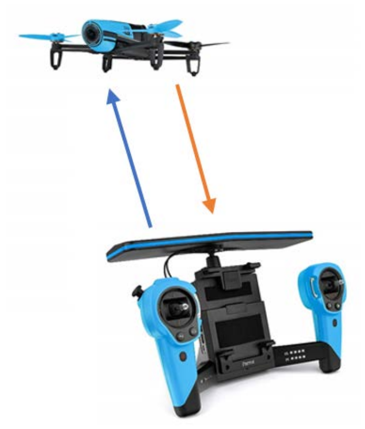
A. INTRODUCTION TO THE COMMERCIAL-OFF-THE-SHELF QUADCOPTER

The choice of the COTSQ that was used for this experiment was not a trivial one. Rapid advancements of microcontrollers, miniature sensors, and actuators resulted in a big variety of inexpensive RC quadrotors on the commercial market. Achieving autonomous flight with a COTSQ requires the ability to communicate control commands from a "computer" in real time with a high update rate to achieve flight agility not achievable by human pilots. Another deciding factor was the ability to integrate a quadcopter with an indoor positioning system such as the Vicon Motion Capture System (VMCS) to provide state feed back of the COTSQ in an indoor environment. guarantee safety of flight during the experiment. The quadcopter was chosen due to its method of communication with the transmitter and the established ability to utilize the Robot Operation System (ROS). This communication established a means of interacting with the low-level controls of the COTSQ. The quadcopter chosen was the Bebop by Parrot Inc. This quadcopter has GPS, accelerometers and magnetometers that are integrated by the Inertial Measurement Unit (IMU). It also has a 14 mega pixel 1080p HD camera with 3 axes digital stabilization [5]. The Parrot Bebop quadcopter also has Visual Inertial Odometry (VIO) algorithms implemented onboard. VIO uses a downward facing camera to compare features within images with measurements from the IMU to provide the odometry measurements and localize the quadcopter within a GPS-denied environment [6].

B. COMMUNICATION STRUCTURE FOR COTSQ

The Parrot Bebop quadcopter utilizes a 2.4 GHz or 5 GHz Wi-Fi connection for telemetry data and control link [5]. The network connection is established between the Parrot Bebop Quadcopter and either the Parrot Bebop Controller or a smart device running the FreeFlight application [5]. In this configuration, the Bebop acts as the network host while the controller or smartphone act as the network client. The user can send roll, pitch, yaw and altitude commands in order to control the flight attitude; these commands are sent via the proprietary communication structure. See Appendix A for technical information on Bebop communications.

All the Parrot Bebop quadcopters are delivered with the same 192.168.42.1 default IP address [5]. In the stock configuration, no more than one quadcopter can be connected to the same controller. User commands given through the control channel are then packaged via User Defined Protocol (UDP) and transmitted to the quadcopter where they are unpackaged and executed by the flight control system. The sensor data (such as GPS, velocity, acceleration, altitude and video feed) is passed to the user and the control commands are sent back to the onboard autopilot via the same UDP communication. Therefore, the only components in the communication/navigation framework are the quadcopter identified by the IP address and the controller. The overall default communication structure can be seen in Figure 1.



The blue arrow represents user commands and the red arrow represents sensor data passed to the operator

Figure 1. COTSQ communication structure. Adapted from [7].

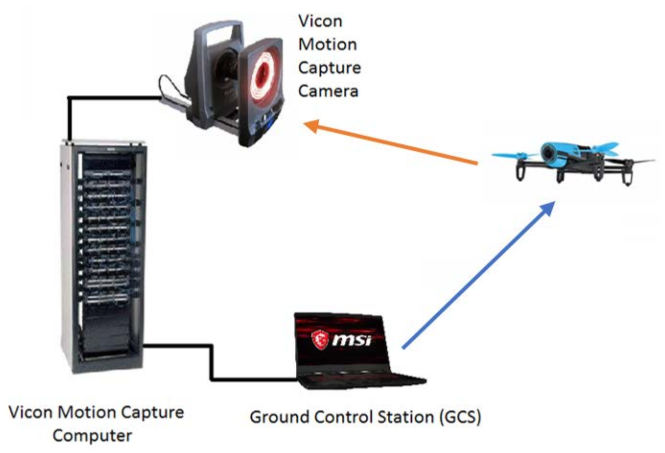
C. COMMUNICATION STRUCTURE FOR AUTONOMOUS CONTROL

In order to enable automatic control of the unmanned aerial vehicle (UAV), the stock IP address is changed to 192.168.101.1 through 192.168.103.1. The onboard software change did not require re-flashing of the Bebop firmware. The IP address change was done so that each of the multiple (three in the scope of this work) quadcopters would act as a host on their own subnet, see more details on the configuration in Appendix A. This then allows one control algorithm executed on a single PC to connect to all three subnets (3 quadcopters) and send individual commands to each quadcopter. If the IP addresses were in the same subnet, then the commands for the first quadcopter would be executed by all the quadcopters in that subnet and operators would not be able to send individual commands.

The custom-built control algorithm is executed on a single PC that represents a GCS. The GCS runs a ROS operating system in Linux [8]. For this GCS to communicate with the quadcopter, the UDP packets must match what the quadcopter receives from the controller. This is done with a Linux SDK package called Bebop_Autonomy that changes the ROS topics into the communication form that the quadcopter can recognize.

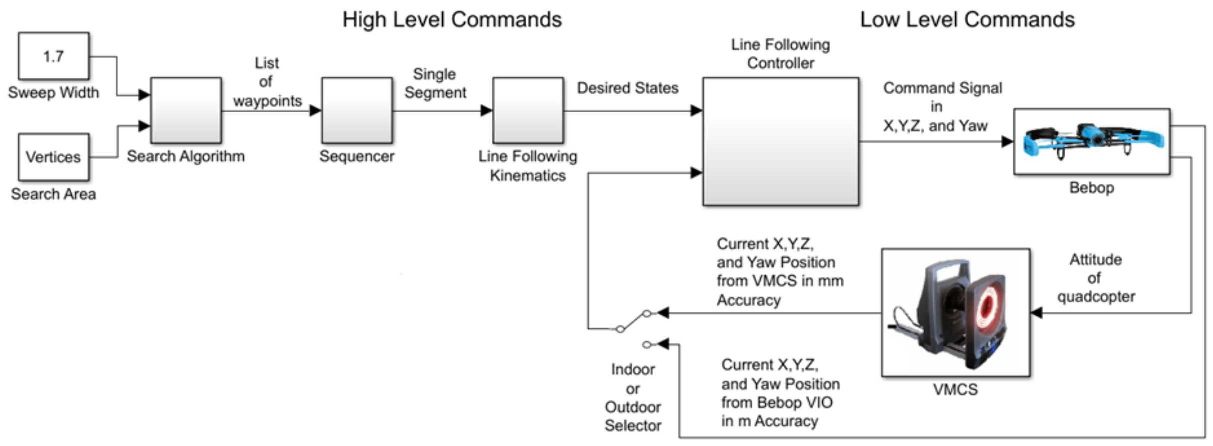
The GCS brings all the sensor data in and processes it to send roll, pitch, yaw, and altitude commands to the quadcopter. The sensor data that come into the GCS for this experiment include position and attitude data from a VMCS and video stream from the quadcopter. Due to the high bandwidth requirements of the video link, the IMU data from the Bebop is updated at 5 Hz. The VMCS system and the position data from the VIO on the Bebop are updated at 100 Hz. If the quadcopter is flown outside, the VMCS is replaced with the onboard GPS and VIO. The controller is discussed below but is designed to transition seamlessly between VMCS and VIO feedback.

In the indoor configuration, the attitude of the quadcopter is captured by the VMCS, this data is processed by the VMCS computer and sent to the GCS. Once in the GCS, the data is processed by the custom-built high-level flight controller and the corresponding low-level commands are sent to the quadcopter. These commands are processed by the quadcopter and executed as attitude changes which are then read by the VMCS again. This provides the feedback loop of all states for the experiment. This whole communication path can be seen in Figure 2. The data that is passed between each controller is shown in Figure 3. The components in Figure 3 will be expanded upon in Chapter III.



Vicon Motion Capture Camera
Vicon Motion Capture Computer
Ground Control Station (GCS)

Figure 2. Communication structure for indoor autonomous flight. Adapted from [7].



High Level Commands
Low Level Commands
1.7
Sweep Width
Vertices
Search Area
Search Algorithm
List of waypoints
Sequencer
Single Segment
Line Following Kinematics
Desired States
Line Following Controller
Command Signal in X,Y,Z, and Yaw
Bebop
Attitude of quadcopter
VMCS
Current X,Y,Z, and Yaw Position from VMCS in mm Accuracy
Current X,Y,Z, and Yaw Position from Bebop VIO in m Accuracy
Indoor or Outdoor Selector

Figure 3. Data passed between each part of the control architecture

D. ESSENTIALS OF THE SIX DEGREES OF FREEDOM DYNAMICS OF QUADCOPTERS

Multicopters are UAVs that use multiple propellers to move in rotation and translation in space. These multicopters can be in many different configurations, but typically will be symmetric and have an even number of propellers; these are the baseline characteristics of most simple designs. The propellers themselves can have two or more blades. Movement of the multicopter is accomplished by speeding up or slowing down pairs of the multicopter propellers. The four-propeller case with X configuration will be discussed from this point forward. This configuration can be seen in Figure 4.

In quadcopter flight, each propeller only provides thrust in the body fixed z-direction. Also, the only input to each of the propellers is the motor speed. Thus, the propeller speed can be increased or decreased while in flight which increases or decreases the thrust from the propeller. The sum of

$$F_i = K_f \times w_i^2 \tag{1}$$

for all four motors produces the lift and translational for the quadcopter [9]. In (2), ω is the angular velocity of the propeller and K_f is the aerodynamic constant in the vertical direction [9]. The

$$M_i = K_m \times w_i^2 \tag{2}$$

is caused by the drag due to air resistance. In (2), ω is the angular velocity of the propeller and K_m is the aerodynamic constant in the horizontal direction. The maximum torque on the motor is limited by (2) [9].

In order to get motion in any direction other than the z-axis, the total force produced by the rotors needs to be rotated in the desired direction of travel. Rotating the UAV body with multiple rotors spinning requires a controlled mismatch in propeller speed. This mismatch can cause a torque on the body or a force on the body depending on which pairs of propeller blades are sped up and slowed down. Torques will cause the body to rotate about its center of gravity while a force will cause the body to translate. The

$$F_T = F_1 + F_2 + F_3 + F_4 - mg \tag{3}$$

in the Z direction cause the quadcopter to ascend, descend or hover [9]. If (3) equals zero, then the quadcopter will hover. If (3) is greater than zero, then the quadcopter will climb and if (3) is less than zero then the quadcopter will descend.

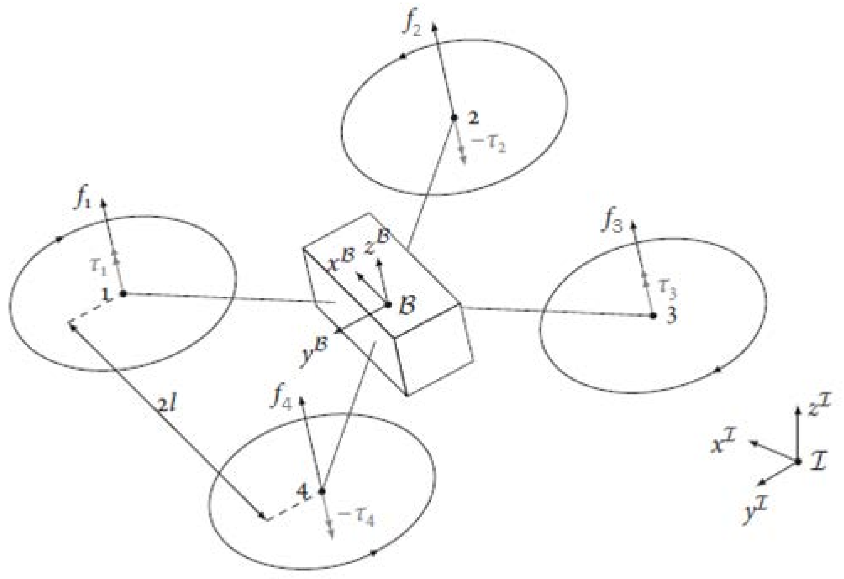


Figure 4. Freebody Diagram of X quadcopter. Adapted from [10].

Once the quadcopter has a roll or pitch angle (via controlled differential of the RPM of opposite rotors) it will tilt in the direction of travel because a portion of the total (1) is projected into X and Y directions, thus causing lateral motion in that direction. In turn, this tilt angle takes some of the thrust out of the z-axis and the quadcopter will sink. In order to maintain level flight, the speed of the motors must be increased to compensate for the same amount of lift as before. The

$$T_x = (((F_1 + F_4) - (F_2 + F_3)) * L) \tag{4}$$

and

$$T_y = (((F_1 + F_2) - (F_3 + F_4)) * L) \tag{5}$$

cause the quadcopter to pitch and roll [9]. L in (4) and (5) is the half distance between propellers.

When the quadcopter is moving laterally and the operator needs it to stop, the pitching or rolling angle must be fully reversed for the quadcopter to slow translation and eventually come to a stop. This is accomplished by reversing the high thrust and low thrust motors in (4) and (5).

E. SYSTEM IDENTIFICATION

When using a commercial-off-the-shelf quadcopter with a proprietary embedded inner loop (that also does not allow any modification of its architecture other than tunable parameters that affect flight agility), it is difficult to know what control methods are being employed and how quickly the inner loop will react to commands without prior testing. If the inner loop control laws were known the system response would be able to predict the performance of the inner loop. Transient response characteristics of the quadcopter define the dynamics of states of the visual sensor that is rigidly attached to the platform. In many cases the Line of Sight (LOS) of the sensor can be pointed, however, it is not gyro-stabilized in inertial space. Considering the application of the sensor to the task of visual search, the transient characteristics of the LOS are essential in building the robust detection capability. More details about the video sensor and detection criteria are given in Section G below. Ultimately if the relative velocity between sensor and target is too high, the video sensor could miss the target due to minimum requirements need to constitute a detection.

As previously discussed, the user controls the quadcopter with a transmitter via a Wi-Fi link. The onboard autopilot simplifies the control task for a human by decoupling the attitude dynamics of angular motion. This so-called augmentation mode is typical for fly-by-wire systems. Decoupling control channels mean that the operator can give separate commands and the autopilot holds the other states constant. The four actions controlled by the operator are the roll and pitch angle, the yaw rate, and the altitude rate. These commands are broken into individual channels so that the human operator can physically control the quadcopter. This decoupling of control channels on the quadcopter can be assumed because coupling on multiple channels would complicate the controls to the point that the quadcopter would not be able to be flown by an operator. The GCS is going to utilize the four decoupled channels for control. The system identification testing was utilized to establish a baseline for the inner loop functionality and control method.

1. Control Hierarchy

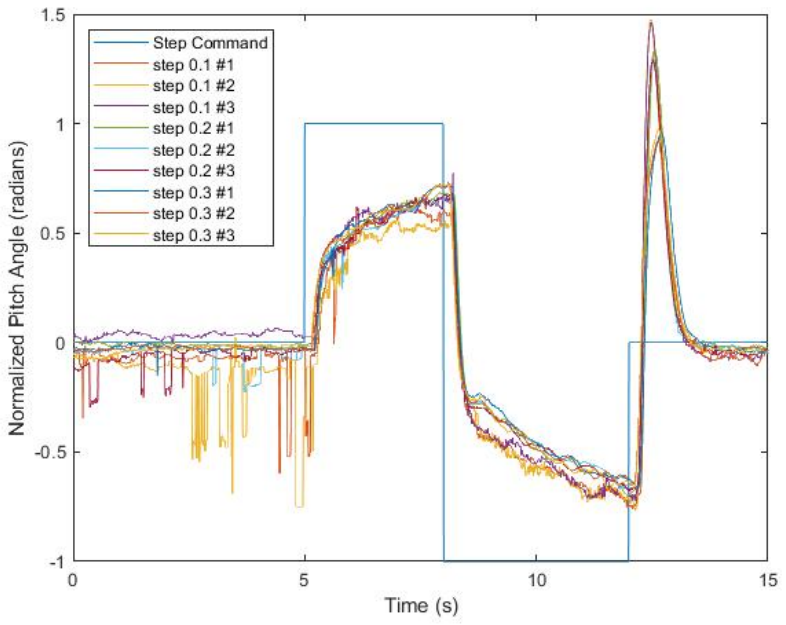
The control architecture of a typical multicopter consists of inner-loop autopilot and the outer-loop controller. In manual RC mode the outer-loop control is implemented by a human pilot that uses visual perception of the flight to solve the guidance and navigation tasks. In automatic mode, the GCS computer utilizes state measurements from VMCS and onboard IMU to essentially solve the same guidance and navigation tasks by the automatic control algorithms. The inner loop controller takes in low-level commands of roll, pitch, yaw and altitude and mixes these commands in such a way as to output individual pulse width modulation (PWM) signals that are then turned into motor revolutions per minute (RPMs). The inner loop has its own feedback loop so that the RPMSs that the motors are spinning at are the RPMs that the inner loop is commanding. The outer loop controller is what provides the roll, pitch, yaw and altitude reference commands to the inner loop. The outer loop itself takes in global X, Y, Z and Yaw coordinates and converts them into the specific roll, pitch, yaw and altitude commands. The outer loop also has a feedback loop that ensures that the current roll, pitch, yaw and altitude states follow the desired commands.

The architecture of the developed outer-loop controller includes the path generation and the path following control algorithms. The path planner solves the mission level task by providing a continuous path as a sequence of straight-line segments; this is a baseline primitive that we adopt in the project. The path planner is combined with a logical sequencer that "discretizes" the path by a sequence of waypoints; its objective is to produce a sequence of two WPs at each instance of flight so that the path following algorithm is always defined. The sequencer is implemented as a Stateflow subsystem.

2. Open Loop Data

The first step in the inner loop identification was to perform open loop data collection in response to a set of predefined low-level commands (also known and implemented as doublets); in this experiment, the GCS gave the Bebop a doublet step command in a single control axis either X, Y, Z or Yaw. A doublet is a positive step followed by a double negative step then followed by a positive step again. The doublet causes the quadcopter to return to its initial starting location.

The quadcopter was flown within the operational field of the VMCS which tracked the x, y, z, roll, pitch, and yaw response of the quadcopter after giving the doublet command. Nine tests were performed on each axis broken up into groups of three. Each group was given a different magnitude step command. The step response was tested on all four command axes. The results for the groups of each axis were normalized to a dimensionless input step response of 1.0 and then plotted. The results can be seen in Figures 5–8. For all four control channels, the command is a dimensionless value between -1 and 1. For the X and Y axis the control signal changes the roll and pitch angle. For the altitude command, the accent or decent rate is controlled. When the fourth channel is commanded it results in a yaw rate. For all four control channels during the test, the step in the positive direction was commanded for 3 seconds and in the negative direction for 4 seconds. The uneven doublet was commanded to get the full response of the system and allow for steady state identification in both directions. The uneven doublet also captures the full reversal response from positive to negative direction. For each test at the end of the return step command, the quadcopter is commanded back to zeros in all four-control axis. The zero command in all four axes produced a large transient that was observed in the X and Y axis at the 16 second mark. The large transient is a result of the Bebop's onboard controller essentially applying the brakes to halt all motion in that X and Y axis. The zero command only occurs during the very specific case of when all commands are zeros and does not occur during any flight regimes as the controller will always be sending some nonzero command in one of the axes.



-

Figure 5. X axis normalized step response (input is dimensionless, output is radians)

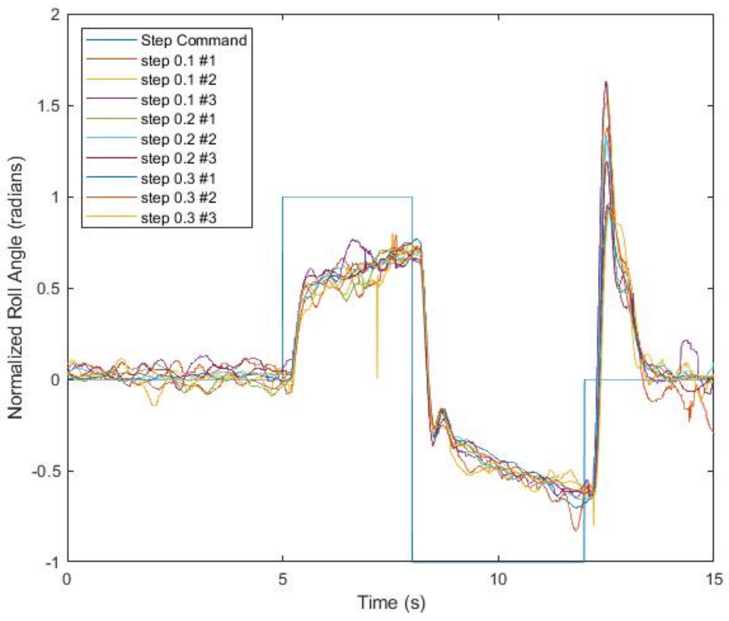


Figure 6. Y axis normalized step response (input is dimensionless, output is radians)

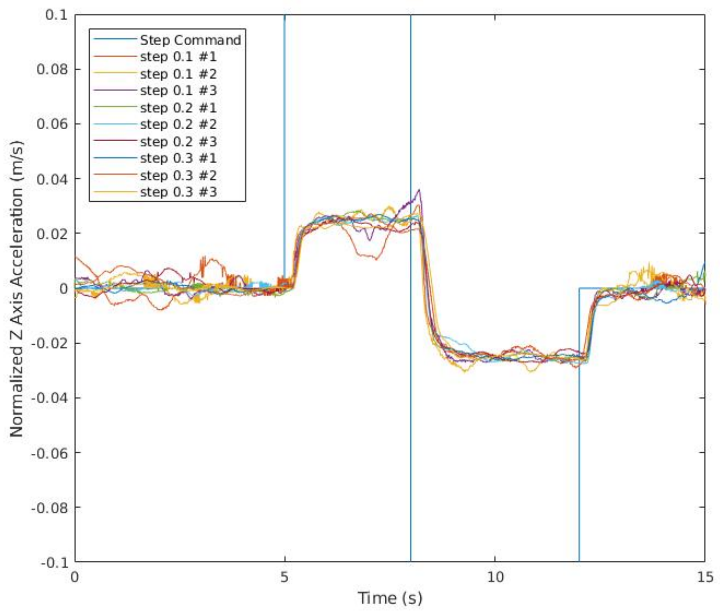


Figure 7. Z axis normalized step response (input is dimensionless, output is m/s)

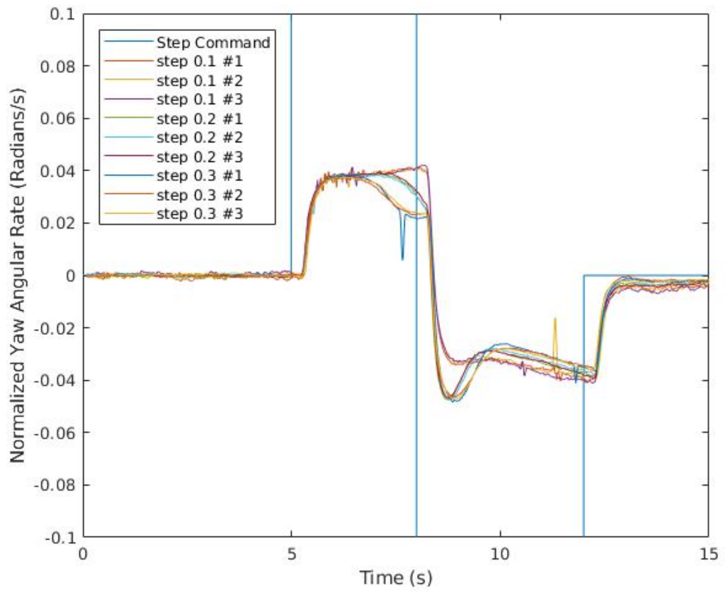


Figure 8. Yaw axis normalized step response (input is dimensionless, output is radians/s)

3. First Order Response Estimation

Once the data was normalized, the process of identifying the system and analyzing the data was completed. The shape of the curve in Figures 5–8 has a first-order response in all four axes. Thus, the general equation for first-order step response was used to model the system. The

$$c(t) = 1 - e^{-at} \tag{6}$$

provides the general step response for a first order system and is in the exponential approach form of the Laplace Equation [11]. The graphical result of (6) can be seen in Figure 9.

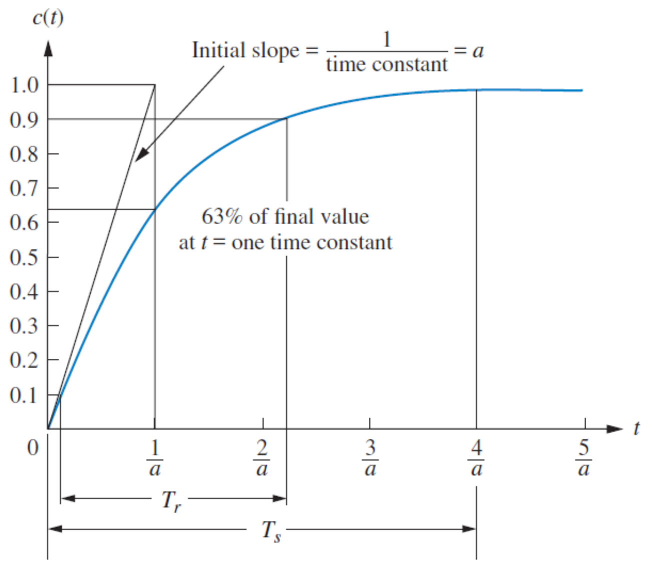


Figure 9. Response for a First Order System. Source: [11].

The first-order system model does not fully match the data. At the beginning of each step response, a delay must be accounted for to fully match the system response. The data was analyzed, and the time delay identified as 0.2 seconds. This delay was measured across all channels and is from communication delays

The delay is the time it took for the controller to create the signal, transmit it to the quadcopter, be processed by the inner loop, execute the command as an attitude change in the quadcopter, be captured and processed by VMCS, transmit the position to the controlling station and be recorded by the controlling station. The delay is only exhibited in the outer-loop controller. The inner-loop delay is assumed to be much smaller due to the numerical execution frequency of the inner-loop. The

$$c_X(s) = 0.65 \cdot \frac{2}{s(s+2)} \cdot e^{-0.2s} \tag{7}$$

shows the x axis estimate for the onboard autopilot controller. The results of (7) are plotted over the recorded data and can be seen in Figure 10. The

$$c_Y(s) = 0.68 \cdot \frac{2.5}{s(s+2.5)} \cdot e^{-0.2s} \tag{8}$$

shows the y axis estimate for the onboard autopilot controller. The results of (8) are plotted over the recorded data and can be seen in Figure 11. The

$$c_Z(s) = 0.025 \cdot \frac{4.54}{s(s+4.54)} \cdot e^{-0.2s} \tag{9}$$

shows the z axis estimate for the onboard autopilot controller. The results of (9) are plotted over the recorded data and can be seen in Figure 12. The

$$c_{YAW}(s) = 0.041 \cdot \frac{3.34}{s(s+3.34)} \cdot e^{-0.2s} \tag{10}$$

shows the yaw estimate for the onboard autopilot controller. The results of (10) are plotted over the recorded data and can be seen in Figure 13. This process of establishing a baseline for the inner loop was proven adequate when the formulated system

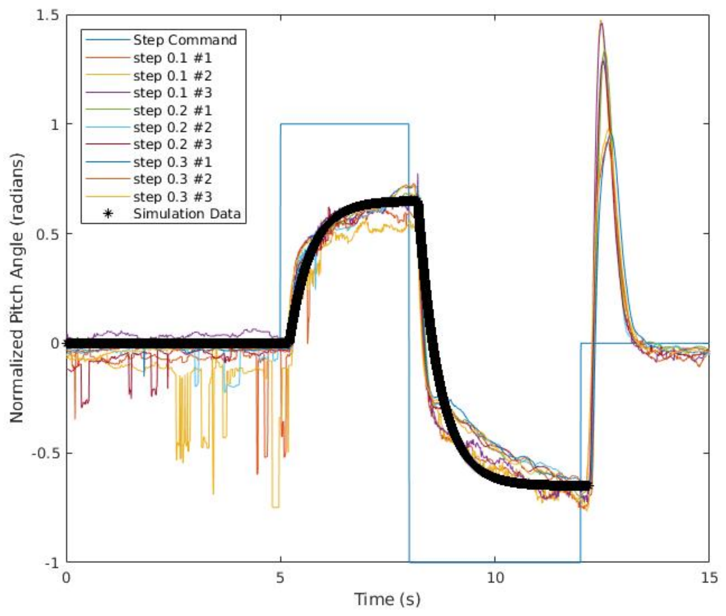


Figure 10. X axis normalized step response (input is dimensionless, output is radians) with system estimate

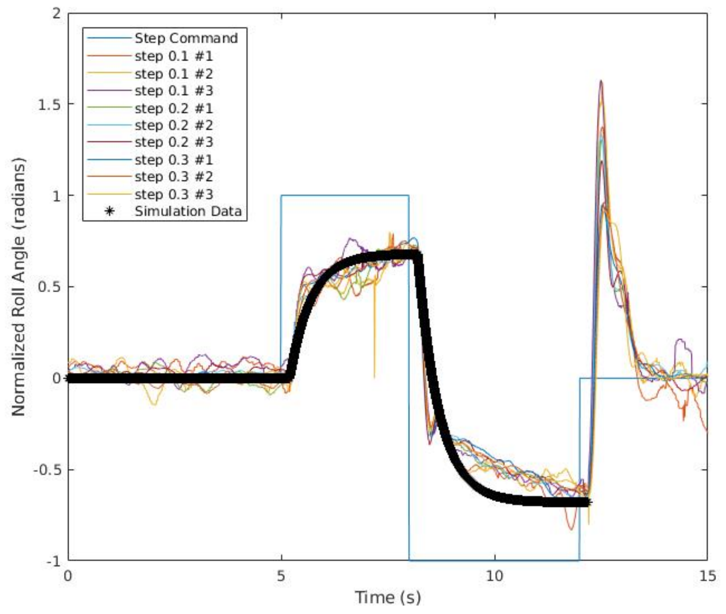


Figure 11. Y axis normalized step response (input is dimensionless, output is radians) with system estimate

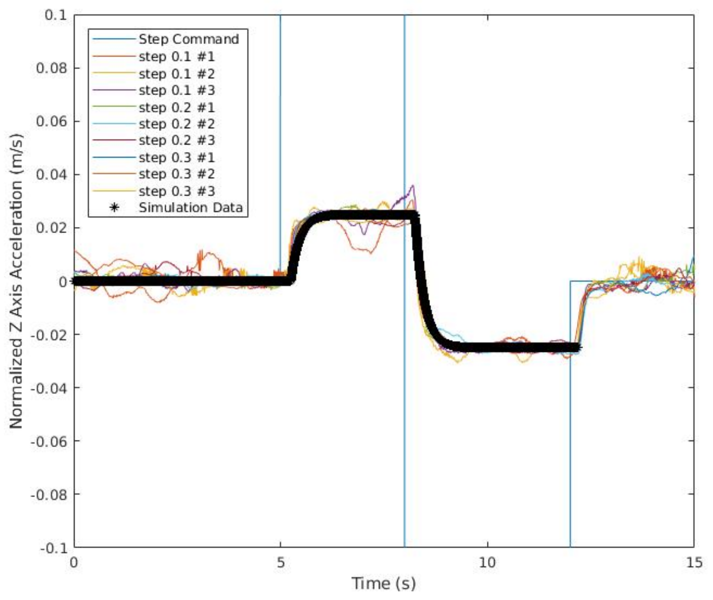


Figure 12. Z axis step response (input is dimensionless, output is m/s) normalized to 1 zoomed into 0.1 with System Estimate

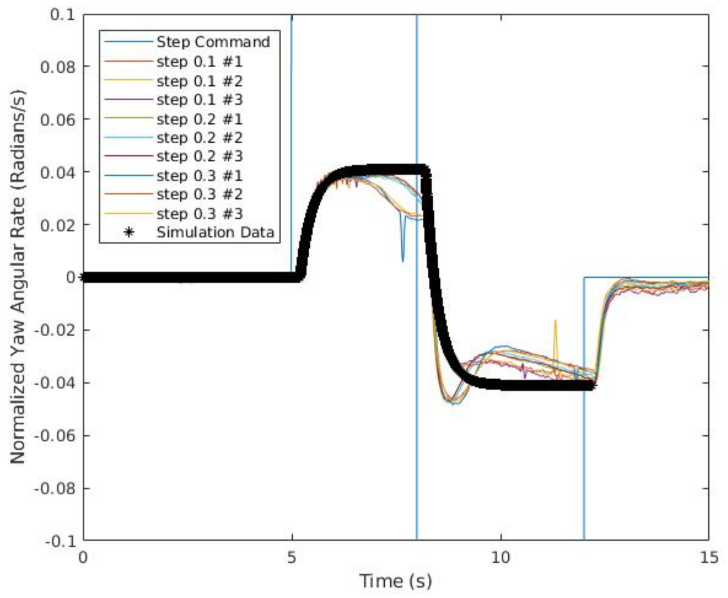


Figure 13. Yaw step response (input is dimensionless, output is m/s) normalized to 1 zoomed into 0.1 with System Estimate

F. COMPUTER VISION BASICS

Computer vision is based on the ability for a computer to use an optical sensor to see the environment it is in. Most modern robots use at least one optical sensor to interact with the world. The prolific use of optical sensors has not always been the case and has really come to the forefront of technology due to the reduced expense of digital cameras. The sensor in the digital camera breaks the captured light into three matrices that represent the color values for Red Green Blue (RGB) [12]. The intensity of each color in the corresponding matrix takes an unsigned integer 8bit (uint8) value from 0–256. When all three of these matrices are combined, a single-color image is formed [12]. These RGB matrices can be converted into other methods of seeing the same image. For instance, the RGB image can be converted by a hue transformation into a Hue Saturation Value (HSV) image [12]. The transformation from RGB to HSV is a change from a Cartesian coordinate system into a Cylindrical coordinate system [12]. Hue images are also comprised of three matrices. The first being the hue itself, the next is the saturation of the color and the last is the brightness of the color [12].

Hue images have a distinct advantage when trying to filter a color that is not just red, green, or blue. In hue images all colors are mapped to 360 degrees [12]. For instance, red is mapped to the degrees between 0 and 60, yellow is mapped to the degrees between 60–120 and so on [12]. All the colors are centered around the Z axis. The saturation and brightness are mapped in linear ranges from 0 to 100%. Saturation is from the center of the cone outward and brightness is along the Z axis [12]. This mapping can be seen in Figure 14.

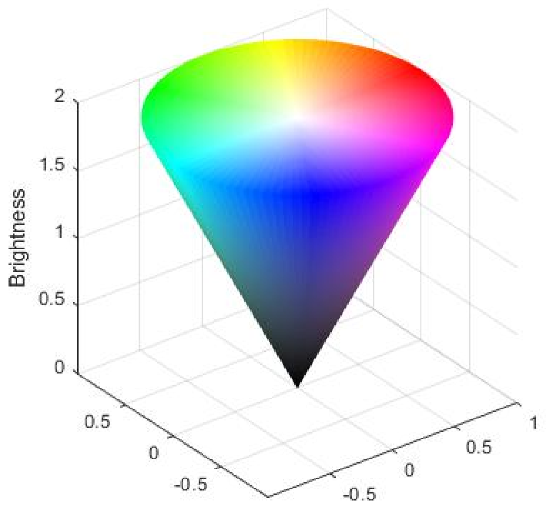


Figure 14. HSV color mapping

Hue images are also less susceptible to the variation of lighting conditions. For instance, in low level light a green tennis ball can look green and with higher levels of light it can look yellow or even white to some sensors. In order to capture the tennis ball in the image, large changes in a RGB filter are required. The only changes required for a hue image is a change in the brightness band. This extensive color mapping pallet allows the programmer to more precisely select colors from an image and apply masks to that image. A mask is an overlay applied to the image that conceals the unwanted colors. The mask typically has an upper and lower limit. Any color within the upper and lower limit are displayed. All other colors are made white or black depending on the mask. This transformation and masking process was made easier with the implementation of Open source Computer Vision software (OpenCV).

OpenCV is an open source software library that can be programmed in C/C++, Python, or Java [13]. This library has a Berkeley Software Development (BSD) license, which allows for open source development. The software is not proprietary and allows for the community that uses it to advance its capabilities. The software can be run on all major operating systems including Microsoft, Mac OS, Linux, iOS, and Android [13]. This library is specifically built to handle computer vision tasks such as color filtering, binary filtering, contour detection, and image recognition [13]. The library can also be used for machine learning through images. Image processing tasks require large processing and RAM capabilities on the platform. OpenCV utilizes the ability to use multicore processing through parallel paths that reduce the processing and RAM requirements [13]. The parallel computing in OpenCV allow smaller devices such as a cell phone or Raspberry Pi to utilize this software library. For further information refer to reference [13]

III. CONTROL SYSTEM DESIGN

The objective of the control system design is to enable autonomous control of the quadcopter, which is typically performed from an onboard or a remote computer, to be included into the sensing and actuation loop of the robot. Refer to Figure 1 and 2 to see the comparison of the human operated and autonomous robots. The full controller that is executed on the GCS takes the search field size and the sweep width of the sensor (field of view of the camera) and converts these parameters into the automatic flight actions of the quadcopter in order to accomplish its mission. There are many algorithms that perform data manipulation taken from the reference commands (user inputs) and the feedback (sensor data) to allow for precise movement of the quadcopter along the complex path.

This chapter starts with the discussion of theory behind moving the quadcopter from one location to another. Next, it presents a model of a “bike” that is used to represent the lateral movement of the quadcopter. The video camera needed to face the direction of flight so that it is imperative to enable the camera directional control while in flight. This chapter also discusses various modes of control for a multicopter, including heading control. Controlling the heading of the quadcopter is the least efficient and slowest responding mode of control due to the flight dynamics explained in Chapter II. Typically, quadcopters are flown with a fixed heading and only roll and pitch are used for lateral movement.

A. PATH PLANNING

There are an infinite number of paths that can be taken to get from one point to another. One method of solving the task is to fly along the straight segments that discretize an arbitrary path in space. This method is outlined in [14]. In this method, a bike is driven by only two primitive commands: the velocity and the steering angle that allows the bike to reach the new point. This method is detailed below. Moving from point to point can be taken one step farther. In the next more complex method, the bike joins the line between the two points and continues along the line until the final point and orientation is reached. This method is also outlined in [14] as well and will be detailed below.

1. Traveling to a Point in Space

In the example that is given in [14], Peter Corke demonstrates that by using the model of a bike in a two-dimensional space, you can start at any point and only by controlling the speed and the steering angle, the bike can be driven to any other point within that space. This model uses proportional control only and introduces the principles of feedback control. In general, the same architecture can be extended to a more general class of controllers such as PID. See Figure 15 for the example.

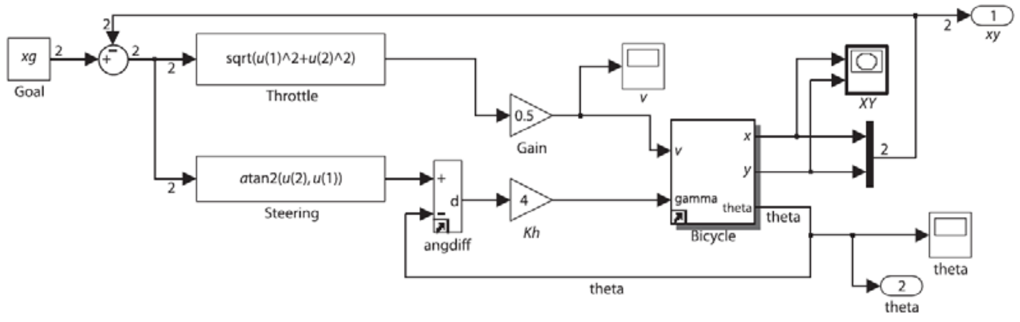


Figure 15. Moving to a point Simulink code. Source: [14].

The essence of Corke's example is that the bike can be turned in a circle until the bike is pointing at the new point, at which time the bike is ridden straight toward the desired end point. The

$$\gamma = K_h * \theta \tag{11}$$

provides the steering command to the bicycle plant [14]. The (11) is comprised of K_h which is the proportional gain that controls the steering angle, and θ is the error between the desired steering angle and the actual steering angle. The

$$v^* = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)} \tag{12}$$

is proportional to the distance from the bike's current location to the desired point [14]. The relation

$$v = K_v * v^* \tag{13}$$

is derived from (12) as suggested by reference [14], where (13) is comprised of K_v, which is the proportional gain that controls the amplitude of the velocity, and v^* is the distance from the current position to the final point. Thus, (13) is faster the farther away the bike is from the desired point and slower the closer it is to the desired point. Eight different cases of the bike riding example can be seen in Figure 16 [14]. Each of the different cases in Figure 16 start at a different location.

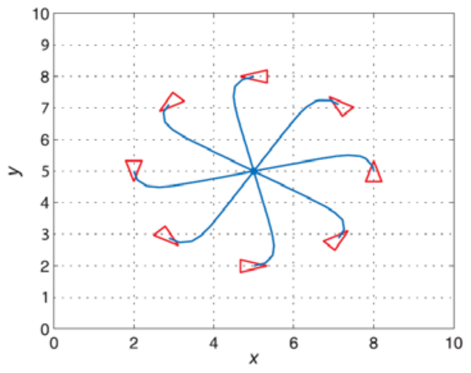


Figure 16. Moving to a point examples. Source: [14].

The problem with moving to a point is that the approach to the point is not controlled and therefore the orientation of the bike at the final point is not controlled. It is important to note that control of the approach angle, i.e., the body heading of the camera is essential for the envisioned search mission. Due to the quadcopter requiring a specific heading at the next waypoint, the approach that only moves the quadcopter to the next point is not sufficient. A better approach is to draw a straight line between the points and then have the bike join the line.

2. Following a Line in Space

In the "Following a Line" example demonstrated in [14], the code utilizes a line provided in standard form and the position of the bike. It then drives the bike to join the line. The algorithm can be seen in Figure 17. This method allows waypoints to be strung together into a continuous path while controlling the orientation of the bike when it reaches the next desired point. In Figure 18, four different starting orientations of the bike are shown that are driven to join the line.

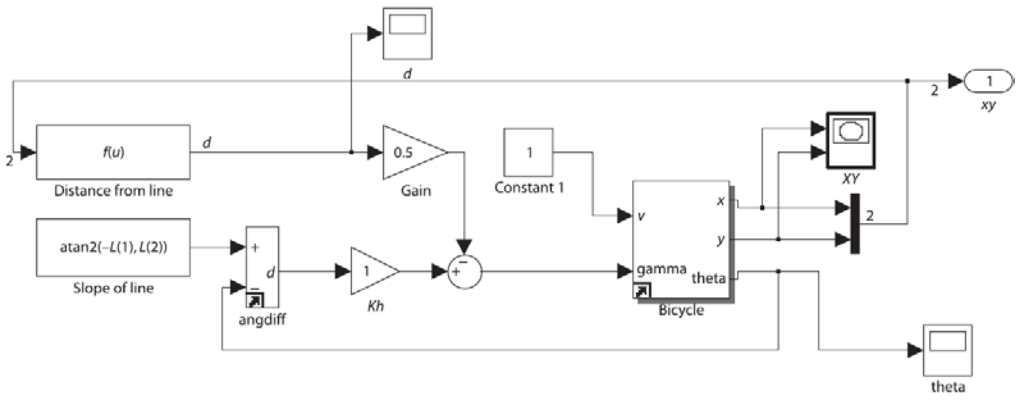


Figure 17. Line following Simulink Code. Source: [14].

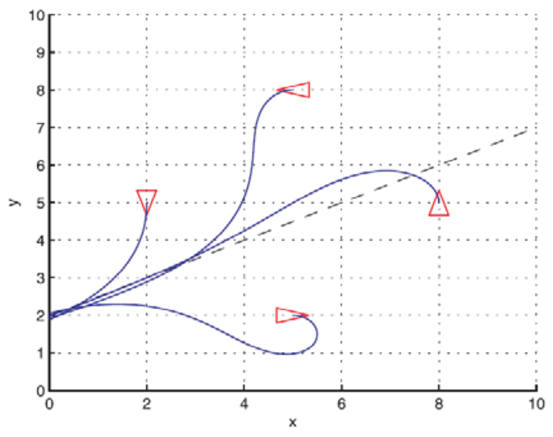


Figure 18. Example of line following. Source: [14].

The algorithm utilizes two errors in the state of the bike relative to the desired line. The first error is the shortest distance from the bike to the line; measured along the line normal to the current location. The next error is the difference between the current heading of the bike and the direction of the desired line. The

$$\gamma = -K_d * d + K_h * (\theta * - \theta) \tag{14}$$

utilizes both the distance error and the heading error to provide the control that drives the bike to the line [14]. In (14), K_d is the proportional gain acting on distance error, d is the normal distance to the line, K_h is the proportional gain acting on the directional error, θ^* is the desired heading, and θ is the current heading [14].

The objective of the controller is to regulate both errors to zero. The distance from the bike to the line will be zero and the heading will be parallel to the line as seen in Figure 18. If the bikes path is parallel to the line and the position of the bike is on the line, then the bike will continue to follow the line. How quickly and smoothly the bike joins the line and continues to follow the line is controlled by two parameters: K_d and K_h. Corke's method provides the basic architecture of the path following controller in two-dimensional space.

B. STATEFLOW BASICS

Developing the ability to fly along a straight line requires an algorithm that generates a new straight segment when the robot finishes the previous one. The implementation of the segment-following approach utilizes a logic-based sequencing algorithm that triggers the transition to the next segment based on the continuous analysis of the distance to the end point of the current segment. In Simulink, the changing of states utilizing logic is best handled with a Stateflow controller. Stateflow is a separate environment in Simulink that allows for sequential decision logic between different states of the machine using flowcharts, state transition tables or truth tables. For more information on Simulink Stateflow refer to [15].

Stateflow is used in the final controller to determine whether the quadcopter is traveling along a line segment or has reached the end of the current line segment and needs to switch to the next line segment. Each line segment is defined by the previous waypoint and the next waypoint in the list. These waypoints will be referred to as the previous waypoint and the desired waypoint for future discussions. The waypoints themselves are imported into the Stateflow machine from the path planning software. More detail about the Stateflow machine and the path planning software is in the corresponding sections below.

C. OVERVIEW OF THE FULL CONTROL SYSTEM INTERCONNECTIONS

This section details the general overview of all parts of the quadcopter controller and provides the details for all main subsections of the overall "mission controller." The highest-level view of the Simulink controller can be seen in Figure 19. The video processor is implemented with a Python code that utilizes the OpenCV library with ROS topic subscription and publishing capability in order to provide the target detection to the full controller. For more details on the video processor see section 6 below.

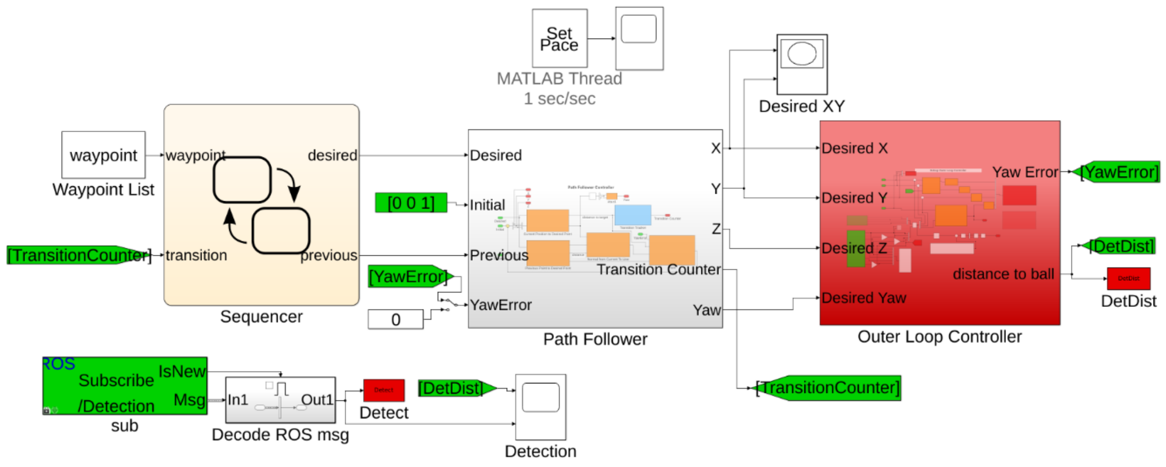


Figure 19. Full Simulink controller

As seen in Figure 19, on the left is the Stateflow sequencer, in the middle is the path following controller and on the right is the outer loop controller. On the bottom left is the video controller detection code. The video detection code reads data from the ROS topic /Detection. The detection data is binary and is provided as either a detect or non-detect of the target. Also, at the top of the code is a set pace block that ensures that the code is running in real time and does not get ahead or behind, as this would "distort" the real time dynamics of the quadcopter.

For the following discussion of each subsystem, the flow will start at the quadcopter with the roll, pitch, yaw and altitude commands and will work backward through the full controller to the generation of waypoints from the path generating code. For easy interpretation of the Simulink code, inputs will be displayed as green blocks, outputs will be red blocks, operations on signals will be displayed in orange and logical operations will be blue.

1. Inner-loop Control

The inner loop for this experiment is housed and operated on the quadcopter. Due to this limitation, the model of the inner loop is not displayed in the Simulink controller. The quadcopter receives four commands in order to fly autonomously. These commands are roll, pitch, yaw and altitude. The system identification that was analyzed in Chapter II resulted in the four equations that described the inner loop response for these four control channels.

The roll, pitch, yaw and altitude commands are calculated by the outer loop controller from the search trajectory discretized by the sequencer and the path follower. The roll, pitch, yaw, and altitude commands are then fed into the Bebop Quadcopter Block. In this block the commands are sent to the quadcopter. The final control commands are converted from Simulink to ROS and then sent via UDP to the Bebop. This communication allows the commands to be read and executed by the quadcopter. As previously discussed, the ROS to UDP transition is handled by the Bebop_Autonomy SDK package and is not pictured. The Bebop quadcopter block that houses the Simulink communication structure can be seen in Figure 20. The Simulink repackaging includes creating a blank ROS message called geometry_msgs/Twist which is a standard message within ROS, populating that message with the commands created by the path following controller and publishing that ROS message to the ROS topic Bebop/cmd_vel for its execution by the inner loop autopilot.

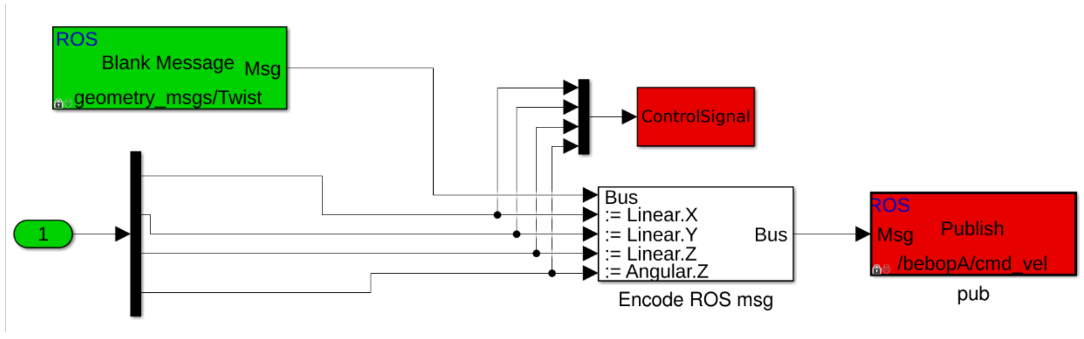


Figure 20. Bebop quadcopter block

2. Outer-loop Implementation

The outer loop of the controller takes in the desired position from the sequencer and outputs the roll, pitch, yaw, and altitude commands to the Bebop Quadcopter block for control signal communication. The Outer Loop controller can be seen in Figure 21. The outer loop uses the feedback from either the GPS and VIO (Feedback_odometry) or the VMCS (Feedback_Vicon). This feedback is used to create an error between the desired attitude and the actual attitude of the quadcopter. For the X, Y, and Z control signals, the error is the difference between the desired and current states.

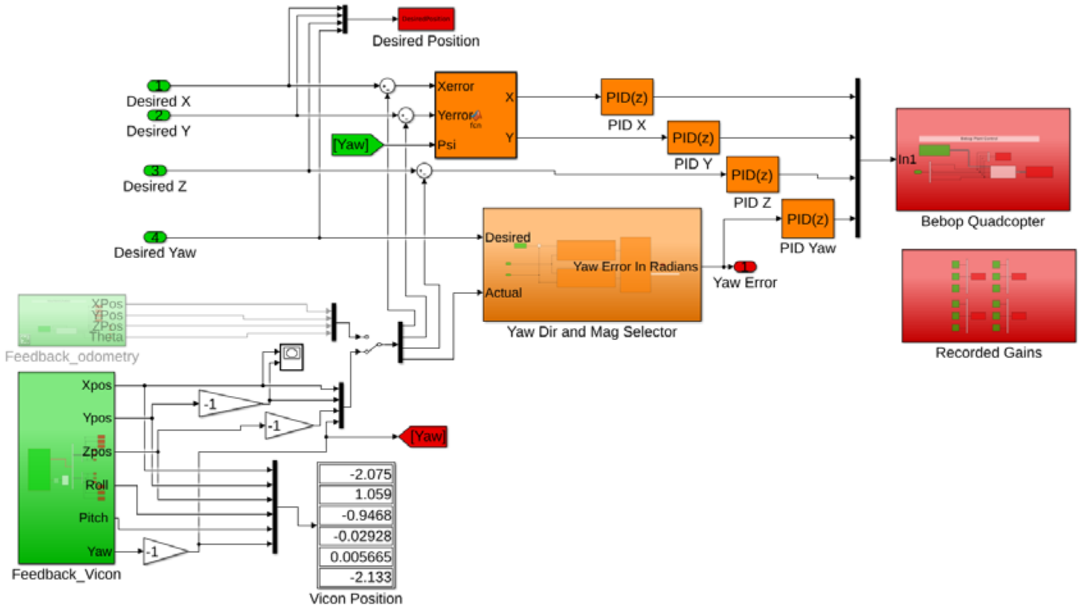


Figure 21. Simulink diagram of the output commands

For the error in yaw, a special logic controller was designed to deal with of heading errors that cross over the boundaries at pi and -pi. This algorithm determines the magnitude of error by using the difference in heading between desired and current and accounts for any heading wrapping that is required. The selector also determines the shortest route between the desired heading and the current heading and provides a direction of rotation either clockwise (CW) or counterclockwise (CCW). The full Yaw Direction and Magnitude Selector can be seen in Figure 22. The errors in X, Y, Z, and Yaw are then fed into a Proportional Integral Derivative (PID) controller, which then outputs the matching commands to the Bebop's inner-loop.

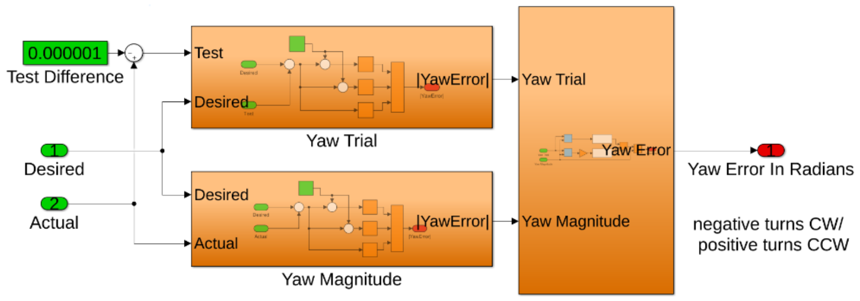


Figure 22. Yaw direction and magnitude selector

PID controllers utilize three gains to provide the correct command to the quadcopter. For a simple example, imagine a car that is trying to merge onto the highway from the on-ramp. The proportional gain takes the current position error from the current position to the desired position and determines the magnitude of the turn needed to drive that error to zero. The Integral gain takes the prior error and tries to drive the prior error between the current position and the desired position to zero. The Derivative gain looks at the future position by analyzing how quickly the car is approaching the desired position and tries to correct the speed of approach so that the car does not miss the desired lane and go through to the other side of the lane or worse the next lane over.

Using the values from the PID calculations, the final control values are then sent to the Bebop Quadcopter block and transmitted to the quadcopter via the communication path detailed above.

3. Path Follower Details

The path follower takes in previous and desired waypoints provided by the stateflow sequencer and creates a reference line between the waypoints. It then simulates an ideal quadcopter that joins the line between the two waypoints, utilizing the initial location of the real quadcopter and transits to the desired waypoint. Once the waypoint is reached, it provides feedback to the Sequencer and the next set of waypoints are calculated. This process is repeated until all the waypoints are visited. The attitude of the ideal quadcopter is output and is used as the desired position for the outer loop controller.

The specific way Bebop accepts control commands requires significant changes to the two-dimensional line following algorithms from [14]. In Corke's approach, he used two errors: one due to the distance to the line and another due to the misalignment with the line. These two errors were derived in different units to come up with a velocity and heading for the bike. This method accomplishes the task of getting a bike to join a line but a better solution for Bebop quadcopters is discussed below. Next, the bike's motion is considered in a two-dimensional plane and the height is considered constant. The quadcopter is dissimilar in this regard and moves in three dimensions, so a three-dimensional solution had to be derived. The Path Follower Controller is a full three-dimensional solution to these issues and can be seen in Figure 23.

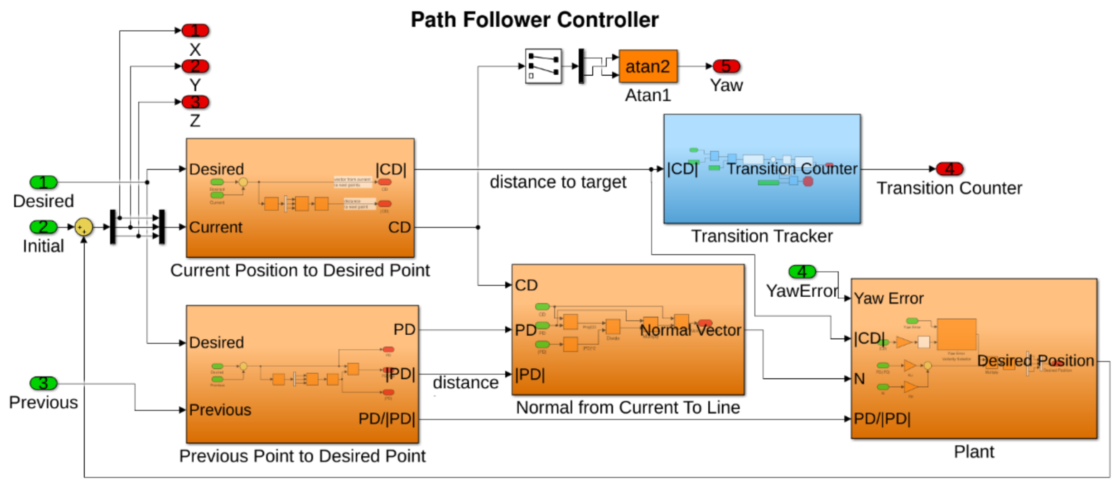


Figure 23. Path follower controller

The main differences between the two-dimensional line follower from [14] and the three-dimensional approach is that all the errors are calculated as vectors. Therefore, they can better represent the errors in three dimensions. To verify that the new vector method was just as adequate as Corke's Line following algorithm, the path follower controller was initially used to drive the bike plant in his code [14]. The new vector approach drives the bike to the line just as well as the two-dimensional approach in [14]. The plotted results of the experiment can be seen in Figure 24. The vector method was then applied to the quadcopter as the new path following algorithm. The first step in detailing the path follower controller is to calculate some basic vectors that are used for calculations.

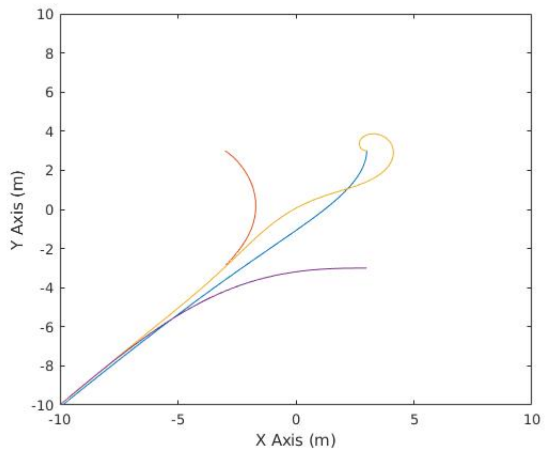


Figure 24. Bike plant is driven by path follower controller

The first vector is from the current position of the quadcopter C to the desired waypoint D. This vector is calculated using the Simulink code in Figure 25 and utilizes the distance method to get the magnitude as well as the vector $\overrightarrow{CD}$ itself. The vector $\overrightarrow{CD}$ will be used in later calculations.

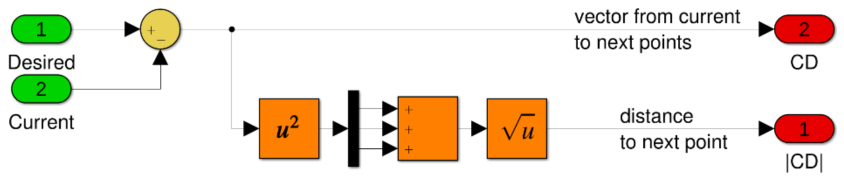


Figure 25. Simulink code that calculates the vector $\overrightarrow{CD}$

The second vector is from the previous waypoint P to the desired waypoint D. The Simulink code that calculates $\overrightarrow{PD}$ can be seen in Figure 26.

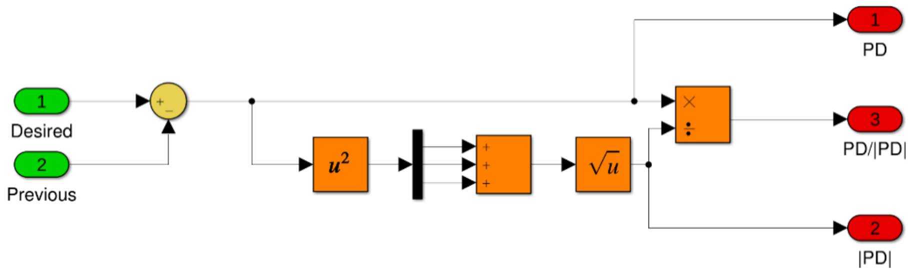


Figure 26. Simulink Code that calculates the vector $\overrightarrow{PD}$

Both vectors $\overrightarrow{CD}$ and $\overrightarrow{PD}$ can be seen in Figure 27 and will be utilized in future calculations of the position errors of the quadcopter. The first error in the bike example is the distance from the current position to the normal of the line. The distance error is replaced with

$$\vec{N} = \overrightarrow{CD} - \frac{\overrightarrow{CD} \cdot \overrightarrow{PD}}{\left|\overrightarrow{PD}\right|} * \hat{u} \qquad (15)$$

that points from the current position of the quadcopter to the line. The graphical representation of (15) can be seen in Figure 27 as the red vector labeled N. The Simulink code that calculates (15) can be seen in Figure 28.

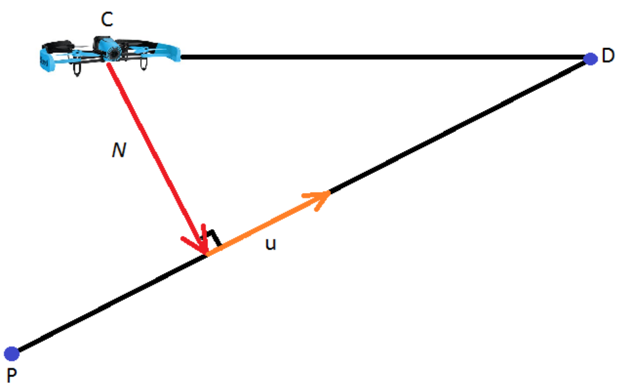


Figure 27. Vector triangle used in path follower controller. Adapted from [7].

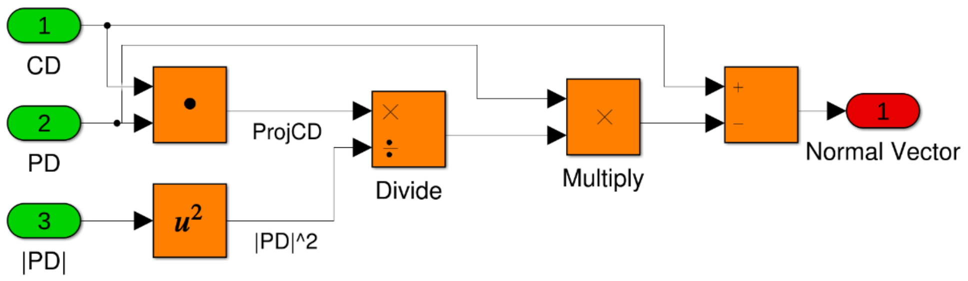


Figure 28. Simulink code that calculates the normal vector $\overrightarrow{N}$ to the line

The second error in the bike example is the heading error where the bike had to use heading to match the direction of the line and drive the error to zero. In the path follower controller, a unit vector is calculated that points along the line for the previous point to the desired point. This unit vector $\hat{u}$ is displayed in orange in Figure 27. These two vectors are scaled by a proportional gain K_N and K_u then added together to achieve a resultant vector. The

$$\overrightarrow{R} = K_N \cdot \overrightarrow{N} + K_u \cdot \hat{u} \tag{16}$$

is then used to control the quadcopter and achieves the same goal of joining the line but in a three-dimensional space.

Depending on whether the line is above or below the quadcopter, the quadcopter will increase or decrease its altitude and maneuver in X and Y to join the line. The quadcopter changing altitude is a significant change to the bike model and allows the full three-dimensional space to be utilized for search.

The next section of the path follower controller is the transition tracker. This section of code takes in the distance from the quadcopter to the desired waypoint and determines if it is within a given threshold. This distance threshold allows the user to define a radius around the desired waypoint that is acceptable to transition to the next line segment in the search pattern. The transition tracker can be seen in Figure 29. The signal is converted from a distance to a binary of 0 or 1. The value of zero means the quadcopter is not within the desired waypoint radius and value one means it is within the desired waypoint radius. This binary value is then recorded into a memory loop where the maximum value is recorded. The maximum value is output as the transition counter and provides feedback to the sequencer. When the transition counter increases by one, the next set of waypoints are provided by the sequencer. When the transition counter exceeds the final waypoint index, the "mission controller" is stopped. Reaching the last transition counter means that the entire search area was exhaustively searched. More information about the search path will be covered further in Chapter IV.

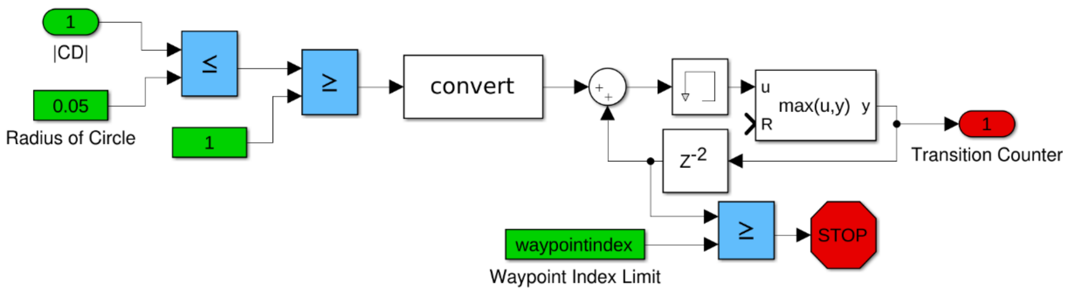


Figure 29. Simulink code for transition tracker

The last part of the path follower controller is the ideal simulation. This Simulink code can be seen in Figure 30. Included in the ideal simulation is the $\overline{R}$ vector from above, the velocity command and the velocity selector. The velocity command is the magnitude of $\overrightarrow{CD}$ which is multiplied by the gain K_v and then limited by a rate limiter.

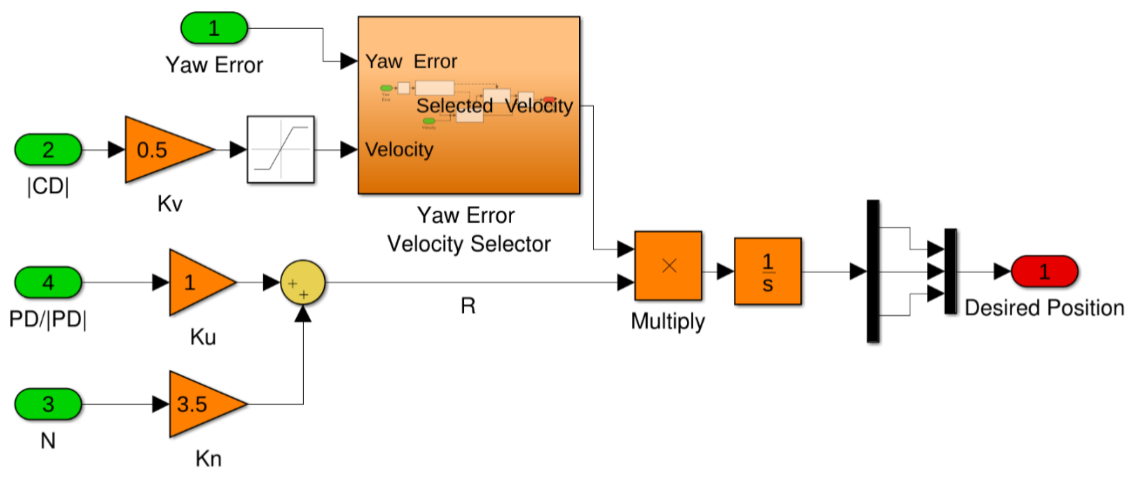


Figure 30. Ideal quadcopter simulation

The yaw-error velocity selector was implemented so that when the heading error between the actual quadcopter and the heading of the ideal quadcopter are greater than 0.174 radians (roughly 10 degrees), the velocity of the ideal quadcopter is set to zero. The yaw-error velocity selector allows the real quadcopter to turn without receiving a large change in velocity and a large heading change at the same time. The Simulink code that executes this logic can be seen in Figure 31.

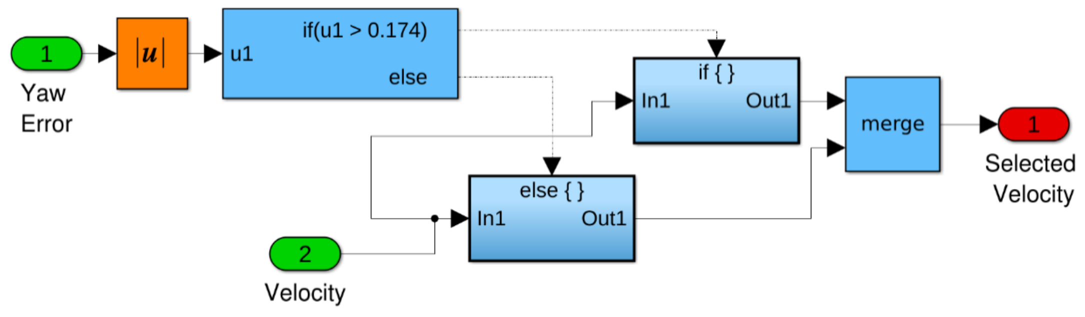
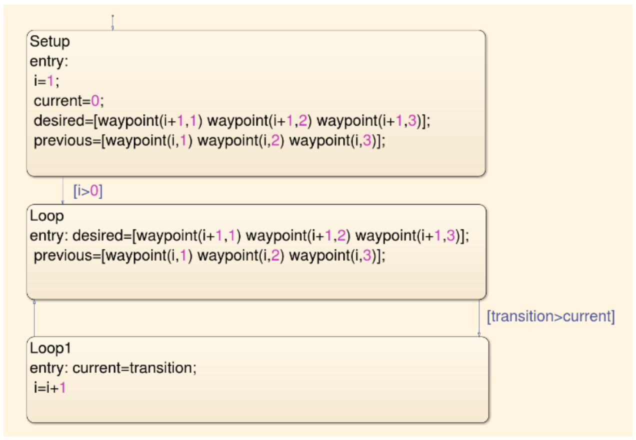


Figure 31. Velocity selector

4. Sequencer Details

The next highest-level controller is the sequencer itself. The sequencer is a Stateflow subsystem that takes a list of waypoints from the path planner algorithm and queues them for the path follower controller. The sequencer can be seen in Figure 32.



```
Setup
entry:
 i=1;
 current=0;
 desired=[waypoint(i+1,1) waypoint(i+1,2) waypoint(i+1,3)];
 previous=[waypoint(i,1) waypoint(i,2) waypoint(i,3)];

[i>0]

Loop
entry: desired=[waypoint(i+1,1) waypoint(i+1,2) waypoint(i+1,3)];
 previous=[waypoint(i,1) waypoint(i,2) waypoint(i,3)];

[transition>current]

Loop1
entry: current=transition;
 i=i+1
```

Figure 32. Sequencer Stateflow controller

In the first block of the sequencer the initial desired and previous waypoints are assigned and the counter i is set to the first waypoint. Once the initial setup is complete, the code then moves to the loop block. The first loop sends the desired and previous waypoints to the path follower. This block will not advance until it gets feedback from the transition tracker in the path follower controller. Once it does receive this feedback, the logic advances to the second loop. In the second loop the current waypoint index is set to the new transition counter and the counter i is increased by one. The second loop then returns control to the first loop. The previous desired waypoint will be the current desired waypoint and the next waypoint in the list will become the desired waypoint. The transitioning of waypoints will continue until the last waypoint becomes the previous waypoint at which time the transition tracker will stop the code.

5. Path Planner Details

The path planner is the highest level of code that is utilized in the "mission controller." It is an algorithm that is run in Matlab in order to provide an exhaustive search of a box. The code takes in the corners of the search box and the sweep width of the sensor and outputs an exhaustive search path in the form of waypoints that ensures complete coverage of the sensor within the search box. More details of exhaustive search and search paths are covered in Chapter IV. The waypoints from this algorithm can then be imported automatically or can be manually transferred by the user.

6. Video Controller Details

Utilizing Simulink for video-processing was costly on processor and memory resources and was ultimately too high for our experiment. A much more efficient method of video processing was to read the ROS topic message from the HD camera directly into a Python script that was utilizing the OpenCV library. The Python code creates a ROS node that subscribed to the ROS topic /Bebop/image_raw. The ROS node then performed a hue transformation on the image. The code applies a color mask to the image and then detects the contour of the target of interest (in this case a tennis ball). Once the contour was defined, the node outputs the color image with contour and text overlays to the user as well as a binary image with the same overlays. For the tennis ball to register as detected, the tennis ball must have a hue, saturation, and brightness that falls between the maximum and minimum values. The tennis ball must also have a pixel area greater than 20 square pixels and it must be detected for more than 7 of the last 14 frames. The detection case of these two images can be seen in Figure 33 and 34. The no detection case of these two images can be seen in Figure 35 and 36

The results of the Python detection software are sent to the "mission controller" as binary data. This binary data represents a detection or no detection case as previously mentioned.

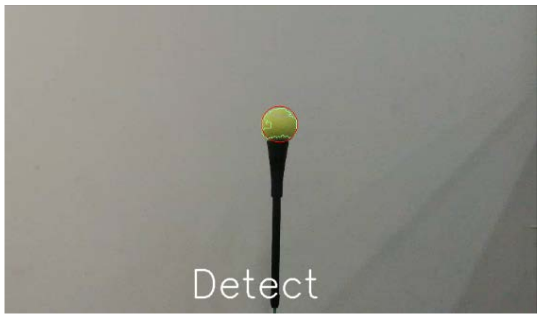


Figure 33. Tennis ball detection in color

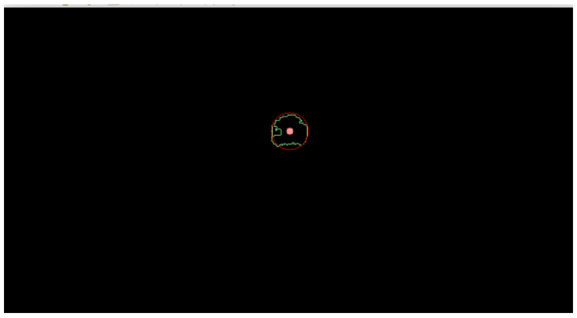


Figure 34. Tennis ball detection in binary

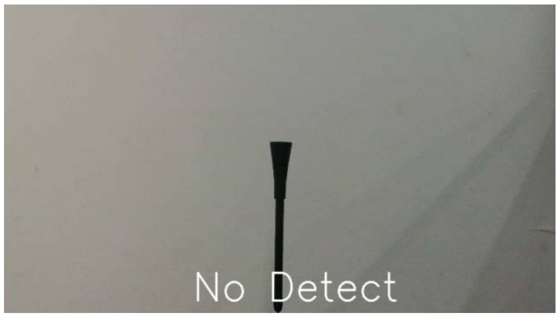


Figure 35. No detection in color

Figure 36. No detection in binary

IV. INTRODUCTION TO SEARCH THEORY

In this section, a basic understanding of search theory will be discussed. The scope of this thesis is focused on utilizing search theory as a tool in order to evaluate the video detector and the flight controller of the quadcopter. Ultimately, the idea of search theory is to have a well-calibrated sensor move through an operational area with the purpose of detecting a target. This general definition has no reliance on the platform that is performing the search, the sensor that is being utilized, nor the type of target that the searcher is trying to find. In actual operational problems, the platform, sensor, and target affect the outcome of the search.

A. SENSOR DEFINITION

A sensor provides a platform with some understanding of the physical world. In search theory, the sensor is a means of detecting a target within the physical world. The sensor can be a touch sensor that informs the platform that it hits a target. It could be a sonar sensor that receives a reflection of an object. It can also be a video feed that utilizes detection software to determine if what is within the field of view is a target or not. This last case is the method utilized in this thesis. The shape and capability of each sensor is different not only between different sensors but also during different environmental conditions. For instance, with the sonar detector, the sea condition can greatly affect the capability for sound propagation through the water. Sea conditions can decrease the ability for the sonar detector to receive the undistorted reflection from an object. For the video feed, lighting conditions can greatly change not only the search parameters but also the outcomes of the target detection. If the lighting conditions are too low, the sensor may not be able to see the object. The best way to deal with these issues is to create a general definition for the sensor.

B. "COOKIE CUTTER" SENSORS

The most general definition of all sensors is a "cookie-cutter" sensor. A "cookie-cutter" sensor is a sensor that has a circular detection shape defined by a radius R [16]. Within the radius R the sensor has a 100% detection rate. Outside of R the detection rate is zero [16]. Most sensors do not have a circular detection but can be converted into an equivalent “cookie cutter” sensor. In order to convert a “non-cookie cutter” sensor into a “cookie cutter” sensor, the sweep width of the sensor must be known. The sweep width is how wide the sensor sweeps out as it is moved through the area and is defined as two times the radius R. One of the ways to determine the sweep width for any sensor is to perform a lateral range curves (LRC) calibration test. This method is only valid if the area being searched is much larger than the sweep width of the sensor [16]. The

$$p(x) = 1 - \exp\left(-\left| \frac{x_0}{x} \right|^b \right) for -\infty \le x \le \infty \tag{17}$$

is symmetric and unimodal and is used to derive the sweep width [16]. The

$$W = \int_{-\infty}^{\infty} p(x)dx \tag{18}$$

utilizing the (17) and is the area under the LRC. The full lateral range curve test for the video sensor will be discussed in section E along with the procedure of its experimental determination. The outcome of the calibration procedure is the probability distribution function that explicitly accounts for the capabilities of a specific sensor. Once the sensor is defined as a “cookie cutter” sensor, the next logical progression is to discuss types of search.

C. BASIC TYPES OF EXHAUSTIVE SEARCHES

Exhaustive search is where every portion of the area to be searched is covered with the “cookie cutter” sensor perfectly without overlap and without going outside the search area [16]. This is the ideal definition of an exhaustive search and is not practical when it comes down to implementing it. For example, consider a square as the simplest shape to be searched. If utilizing a “cookie cutter” sensor which has a round shape, the sensor will never be able to get into the corner perfectly. The mismatch between the corners of the search are and “cookie cutter” sensor leaves areas that are unsearched unless the sensor is brought outside the search area. The mismatch between the round sensor and the search area can be seen in Figure 37.

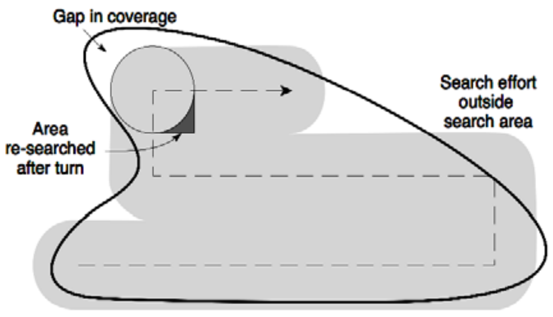

Gap in coverage
Search effort outside search area
Area re-searched after turn

Figure 37. Attempted exhaustive search. Source: [16].

Another example is if there is an object within the search area that causes a split in the search path. The sensor must be picked up from the end of the first search path and moved to the start of the second search path. The movement from one search path to another causes the sensor to overlap previously searched areas. There are several ways of performing an exhaustive search that are not ideal but are feasible and can accomplish different tasks. From search and detection, the first is the boustrophedon method [17], [18]. In this method, the search traverses back and forth across the search area creating a pattern like an ox would plow a field [17]. This method has its advantages because it is easier to create the search path and yet it still covers the search area exhaustively. The second method is the spiral-in method. This method is mostly used to trap moving targets. The third method is the spiral-out method and is better for cases where the target is known to be toward the middle of the search area [16]. In this thesis, we will be utilizing a simple boustrophedon method that is not optimized. Optimizing even a simple boustrophedon method are nondeterministic polynomial-hard problems [17]. The objective of this approach was to design a generalized hardware and software framework that integrated the guidance navigation and control (GNC) algorithms with the search theory methods. Further advancement was made as the key data flow patterns and the fundamental detection methods become integrated.

D. BOUSTROPHEDON SEARCH

The lawn mower algorithm as the vivid representative of the boustrophedon method is commonly referred as a path consisting of straight lines and quick turns [17]. This method lends itself well to computing waypoints for the path planner in Chapter III as each waypoint is the turning points in the lawn mower pattern. This method allows the video sensor to fly in long straight lines, which puts any potential target directly in the flight path of the quadcopter. Also, the turns are quick which allow the camera to sweep through areas that have already been searched by the lateral movement of the quadcopter. The quick turns of the quadcopter reduce the time of researching previously searched areas and increase search efficiency.

E. LRC RESULTS

As previously mentioned, a "non-cookie cutter" sensor can be made equivalent to a "cookie cutter" sensor by performing a LRC test. In order to perform a LRC test, the sensor (in this case, the video camera on the quadcopter) must be "passed" perpendicular to a target at varying lateral ranges. This setup can be seen in Figure 38. The quadcopter traverses along searcher track represented by the black line. In each test the quadcopter traverses this path 10 times in each direction (back and forth) for each position of the tennis ball. The first test starts with the tennis ball directly on the path of the quadcopter. The tennis ball is then moved to the next location along the red arrow. The 10 passes are then repeated along the searchers track. At each location the number of passes and detections are recorded, so that a percentage of detection can be calculated for each lateral range. The lateral range spacing was chosen to be 0.5 m and varied from zero to 2.5 m while the distances was measured with the VMCS. The location of the tennis ball for each LRC test is displayed in Table 1.

Figure 38. LRC test setup

Table 1. Tennis ball location for LRC test

Distance to Centerline (m)	0	0.5	1	1.5	2	2.5
X position (m)	0.010	0.510	0.997	1.498	1.990	2.496
Y position (m)	-3.642	-3.610	-3.602	-3.631	-3.620	-3.616
Z position (m)	0.768	0.767	0.761	0.761	0.758	0.758

The distance from the quadcopter to the tennis ball when it makes a detection is of no consequence for the LRC test. The probabilities of detection from these different tests are used to construct a LRC. The area under the LRC curve is the equivalent sweep width of the "cookie cutter" sensor. The experimental results from the LRC testing can be seen in Table 2.

Table 2. Detection rate of tennis ball for LRC

Distance	0	0.5	1	1.5	2	2.5
Detections	20	20	20	20	17	0
Spurious Detections	0	0	0	0	0	0
Trials	20	20	20	20	20	20
Detection Percentage	100%	100%	100%	100%	85%	0%

The detection results from each experiment can be seen in Figures 39–44. In each of these time history figures, the red line represents the distance that the quadcopter was from the tennis ball at any point within the test. The blue line represents a detection or no detection case. A value of 0 represents no detection and a value of 1 represents a detection of the tennis ball.

Figure 39 displays the centerline test. As expected, this test had the longest duration of detection for the forward and backward translation of the quadcopter. As the tennis ball is moved farther away from the centerline along the lateral line the detection duration was smaller. The smaller detection duration was caused by the tennis ball being closer to the frame of the video and passing out of the video screen quicker than when the ball is directly in front of the camera.

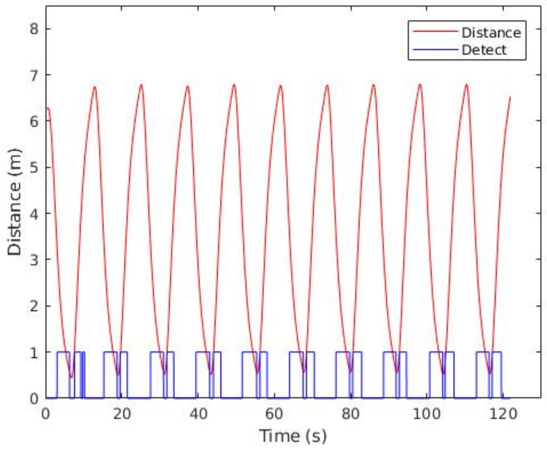


Figure 39. LRC centerline test

As seen in Table 2 and in Figures 39–42, the LRC test from the centerline to the 1.5m lateral distance has a 100% detection rate with zero spurious alarms. These results are a testament to the robustness of the video detection algorithm and the reason for using the hue transformation in the video detection software.

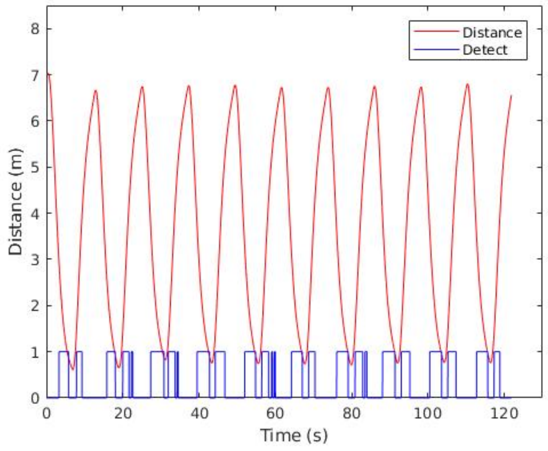


Figure 40. LRC 0.5 m test

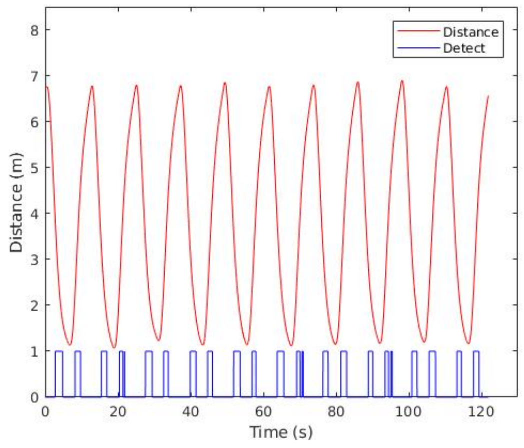


Figure 41. LRC 1 m test

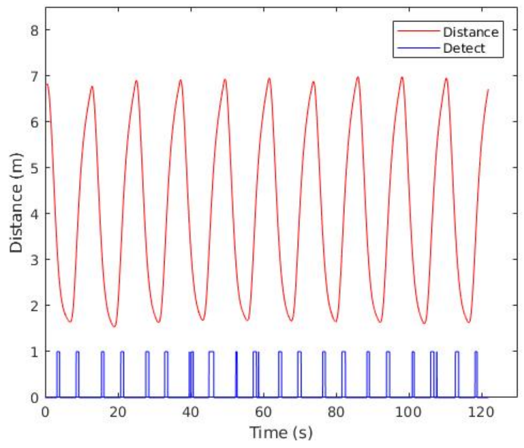


Figure 42. LRC 1.5 m test

The 2 m lateral range test seen in Figure 43 had the first missed detection. The missed detection was due to the ball being on the edge of the video screen. As previously mentioned, for the tennis ball to register as detected the tennis ball must have a hue, saturation, and brightness that falls between the maximum and minimum values, it must have a pixel area greater than 20 square pixels and it must be detected for more than 7 of the last 14 frames. At this lateral range, the angular speed of the ball is higher than before. This increased speed coupled with the ball near the edge of the screen decreased the detection capability.

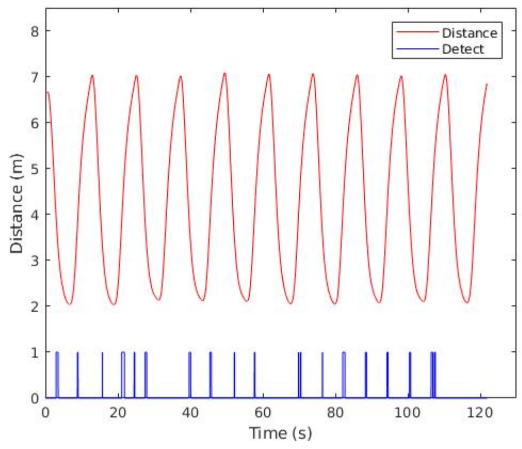


Figure 43. LRC 2 m Test

During the 2.5 m test as seen in Figure 44, no detections were made. The video detector was able to see the tennis ball but did not have enough time to register a full detection. The angular speed of the tennis ball moving through the video frame was so fast that by the time the ball had enough pixel area to meet the minimum 20 square pixels it did not have enough time left in the video frame to meet the minimum 7 out of 14 frame criteria. This result was expected, and the video detection software was not changed as these features also allow not a single spurious detection in the 120 total passes.

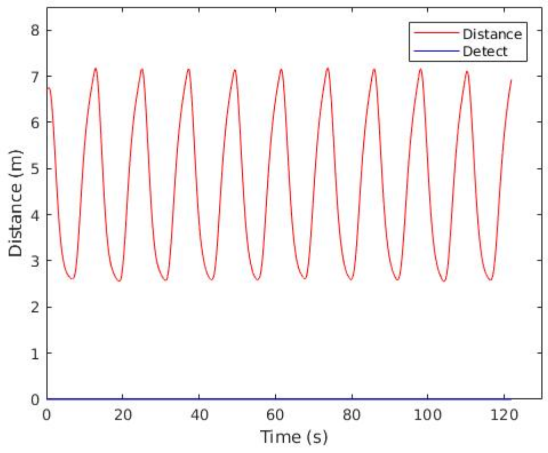


Figure 44. LRC 2.5 m test

The culmination of the LRC tests can be seen in Figure 45 as the overall LRC. In order to achieve 100% detection and to meet the "cookie cutter" sensor criteria, the radius was chosen to equal 1.5 m. The radius of 1.5 m equated to an overall sweep width of 3m for the video detector. Utilizing this sweep width, the path to perform an exhaustive search of a 6m x 6m square is seen in Figure 46.

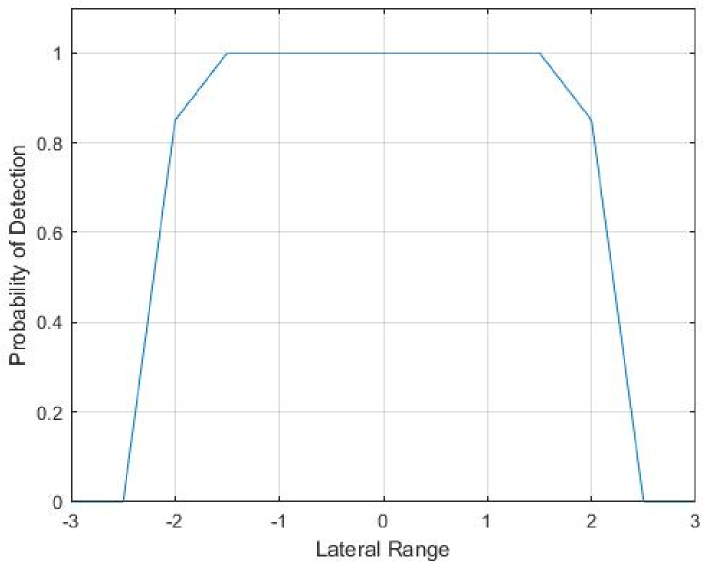


Figure 45. Lateral Range Curve

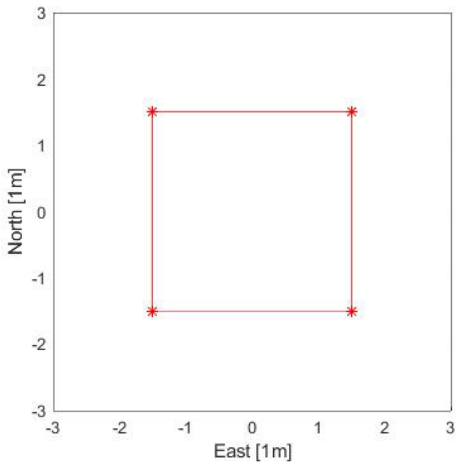


Figure 46. Waypoints for 6 x 6 m search utilizing 3 m sweep width

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION AND TESTING

This chapter covers the key results that allowed for the development of the overall “mission controller” and the video detection algorithm. The first section covers the waypoint tests and addresses the first concern of being able to execute repeatable flight patterns. The next section covers the incremental tests of the video detector. This section addresses the design and robustness of the video detector and its implementation.

The final section reviews the combination of both the motion controller and the video detector into the “mission controller.” This section includes the test results of the previous paths and the final search experiments These experiments highlight the importance of calibrating the video detector and selecting the correct sweep width for the search patterns. A wider sweep width resulted in a less complex search pattern while a narrower sweep width resulted in a more complex search path. The cost of the mission is also directly related to the sweep width.

A. EVALUATION OF MULTISEGMENT PATH FOLLOWING

The quadcopter traversing from one waypoint to the next is the most basic function for the vehicle and is essential to perform its search task. The first test that was performed was the “manual box” test, which is an “open-loop” experiment. In this experiment, the quadrotor executes a sequence of pre-set control commands which allows for robust repeatability of the experiment. This test verified that the quadcopter had the basic functionality to execute the search algorithms in a fully autonomous flight. The next test was an autonomous path following test that implemented the line following and the path following control algorithms. The goal of both of these experiments was to design a controller that offered repeatability when executing different search geometries.

1. Manual Box Test

The “manual box” test proved that the basic communication structure was functioning properly and that all control channels could be controlled by giving basic step commands. These step commands were later replaced by more complex control signals.

The “manual box” test utilized the PID control from the outer loop and manually shifted the desired position over time. The controller would produce individual commands to fly in X, Y, or Yaw. The commands were timed with a delay so that only one command was sent at a time. This method was very rudimentary but was able to provide the proof of concept that the quadcopter could traverse from one desired point to another. The results of this test can be seen in Figure 47.

The “manual box” test had zero regard for the location of the quadcopter at the time of the shift and had to be tuned numerous times to produce a decent box. The zero-feedback shifting of controls meant that the quadcopter would not always make it to the corner of the box or would overshoot before being given the turn command. The error between the actual flight path and the desired path at the corners of the box can be seen in Figure 47. After having this initial proof of concept, the next major progression was to implement the line following controller introduced in Chapter III.

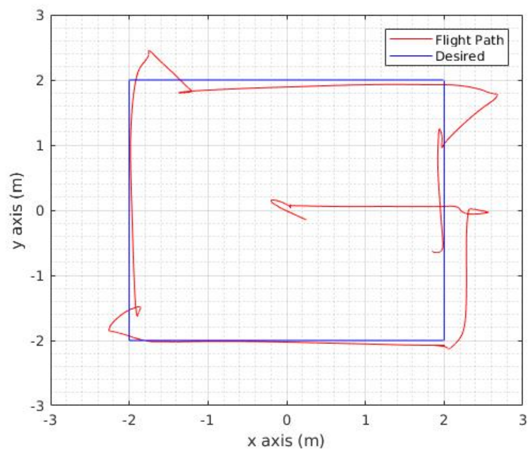


Figure 47. Manual waypoint test verifies basic functionality of the quadcopter

2. Pentagon Test to Prove Path Following

The “pentagon” test validated that the path follower and sequencer controllers were working properly. The path follower allowed the quadcopter to join the lines between sequential waypoints and the sequencer provided the sequenced waypoints. The ability to follow a designed path using only waypoints was the first step in being able to design a search algorithm and execute it.

This “pentagon” test was also the first test to utilize position feedback, waypoint commands, and transition tracking. All these functions are mandatory in successfully accomplishing a search mission. The results of the “pentagon” test can be seen in Figure 48. The pentagon shape was chosen for this test as it maximized the length of each segment and provided multiple turns through all headings. The pentagon shape also allows the video detector to see the entire flight area which test for possible spurious detections. The analysis demonstrates that the addition of the path follower and sequencer controllers allowed the quadcopter to be controlled autonomously by only providing the corner waypoints of the pentagon. Individual commands and timing of the commands no longer had to be calculated and programmed by the operator.

The additional autonomy gained by adding the path follower and sequencer also increased the precision of autonomous search, which was vital to the successful execution of vision-based detection. The best result from the “manual box” test, seen in Figure 47, shows an error between the desired and actual flight path of 0.7 m. In the “pentagon” test, the error between desired and actual flight path was never more than 0.42 m. The “pentagon” test was also performed over a larger area. During the “pentagon” test, the quadcopter averaged less than 0.2 m off the flight path. With the success of the “pentagon” test, all controllers were in place to execute a search path flight. The next major milestone was to develop and test the video detector described in Chapter III

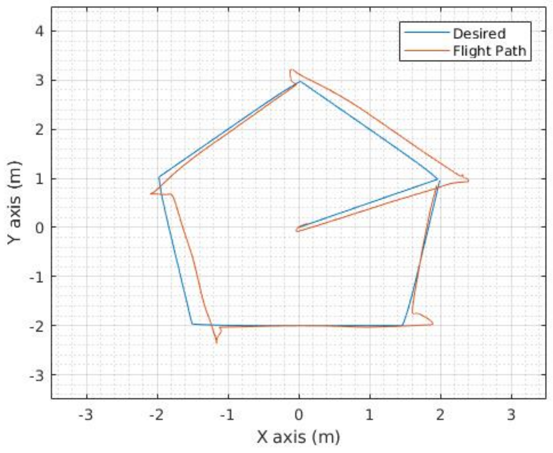


Figure 48. “Pentagon” test desired and actual flight path

B. EVALUATION OF VISION-BASED DETECTION

The detection algorithm is the second half of the overall controller. The sections below detail the tests used to develop and verify the robustness of the vision based detection algorithm. Development of the vision-based detector started out in Simulink and quickly progressed to Python and OpenCV as previously stated. The detection algorithm was developed with a stationary quadcopter and a stationary target as seen in Chapter III. The detector then advanced to a stationary quadcopter and a moving target. The stationary quadcopter and moving target test is described below. Once the detection algorithm was tuned and was successfully working, the next objective was to calibrate the detection software and perform LRC analysis. The LRC testing was detailed in Section E of Chapter IV. Additional to the LRC analysis, a dynamic video detection test was performed that utilized a checkerboard pattern behind the target. The results of that test can be seen in section 2 below.

1. Stationary Quadcopter Test

The “stationary quadcopter” test allowed for the detection algorithm to be fully tested and tuned prior to any flights that relied on the video information. The quadcopter does not have to be in flight for the video to be utilized due to the flexibility of the Bebop_Autonomy SDK package. The first test with the quadcopter and target stationary can be seen in Figures 33–36 in Chapter III. This stationary test provided the convenient framework for the later dynamic tests that were to follow. The next step was to ensure that the frame rate of 30Hz was fast enough to be able to catch moving objects and successfully track them.

During the "stationary quadcopter" test, the target was moved through the video frame to ensure that the quadcopters video feed had sufficient resolution and frame speed to detect the target. The results of one of these tests can be seen in Figures 49–52, where the ball was moved from the right side of the screen to the left side of the screen. Figure 49 shows the detection mask in yellow around the values that match the upper and lower mask for the tennis ball that represented the target. In Figure 49, a red circle encircles the yellow detection blob. The center of this circle is utilized as the centroid for the tennis ball tracking algorithm.

It can be seen in Figures 50–52 that the detection blob changes shape as the lighting conditions and shadow move with the ball. The diameter of the red circle also changes to encircle the detection blob. The centroid and the known diameter of the red circle can be utilized to estimate the distance from the target to the camera.

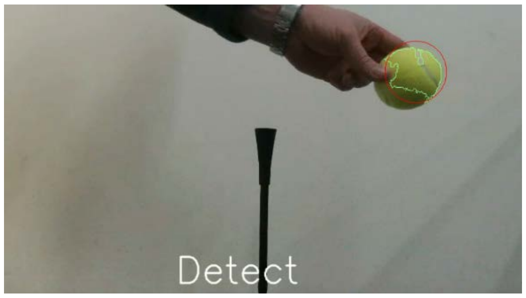


Figure 49. Dynamic "detection test" target far right

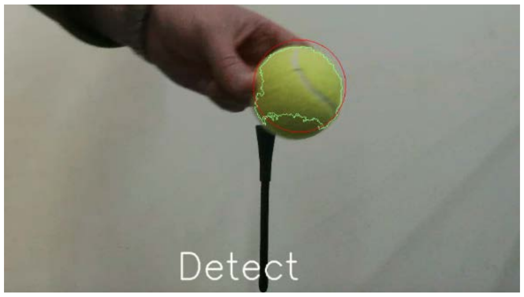


Figure 50. Dynamic “detection test” target centered

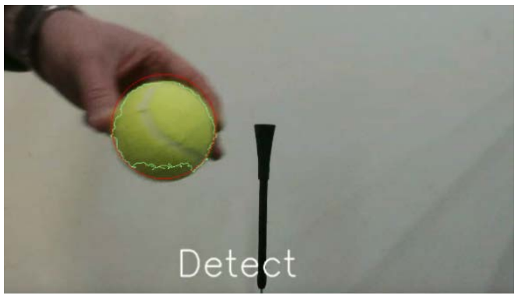


Figure 51. Dynamic “detection test” target left

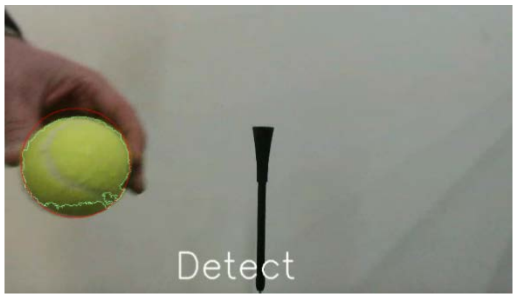


Figure 52. Dynamic “detection test” target far left

2. Detection Algorithm Robustness Test

During the “robustness” test, the objective was to prove that the detection algorithm was robust enough to detect the target in front of a widely varying background. The background varied in color, features, shading, patterns, and lighting conditions. Having a robust detection algorithm that does not provide spurious detections is essential for the success of the overall search experiment. The setup for this test was the same as the LRC test in Section E of Chapter IV except that instead of a neutral background for calibration there were many similarly colored objects in the background. Additionally, a checkboard pattern was placed directly behind the tennis ball. The addition of the checkerboard was an attempt to excite spurious detection either on the checkboard or the background. The results of this test can be seen in Figures 53–55. Figures 53–55 are zoomed in to 150% of the actual image size and encompass the detect/no-detect text and the target in front of the checkerboard. The detection algorithm has two major criteria for detect. The tennis ball must have a pixel area greater than 20 square pixels and it must be detected for more than 7 of the last 14 consecutive frames. Figure 53 shows the frame from the video test prior to the color or pixel size meeting the minimum requirements.

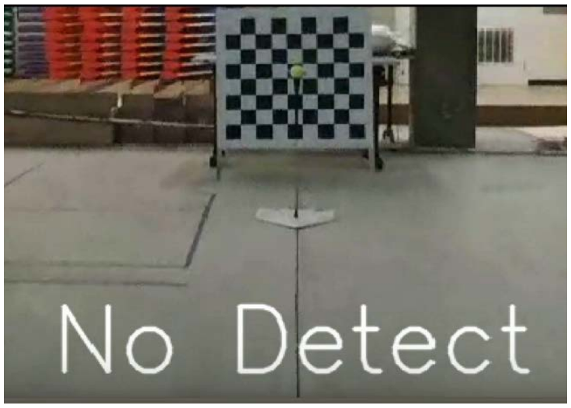


Figure 53. Frame prior to detection in the “robustness test”

Figure 54 shows the frame when the detection algorithm sees the target and the target is greater then 20 square pixels. This frame has not registered as a detect, as it did not meet the 7 out of the last 14 frames requirement of the detection algorithm.

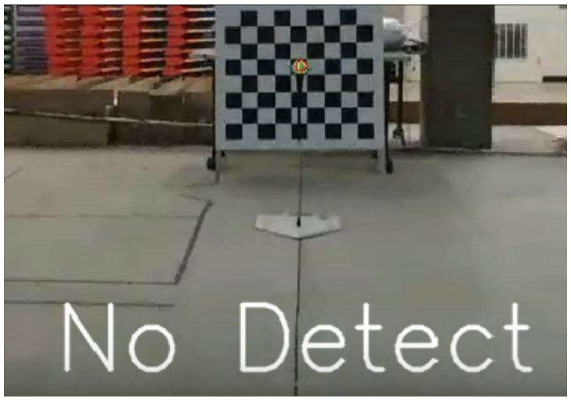


Figure 54. Initial object detection frame prior to average being met

Figure 55 shows the full detection criteria met including the hue mask, object size and average number of frames detected. During this test the only object that was detected was the tennis ball. The checker board had no effect on the detection algorithm that clearly illustrates high robustness of the algorithm. Likewise, the background that has many similarly colored and shaped objects did not produce a spurious detection. The full image can be seen in Figure 56.

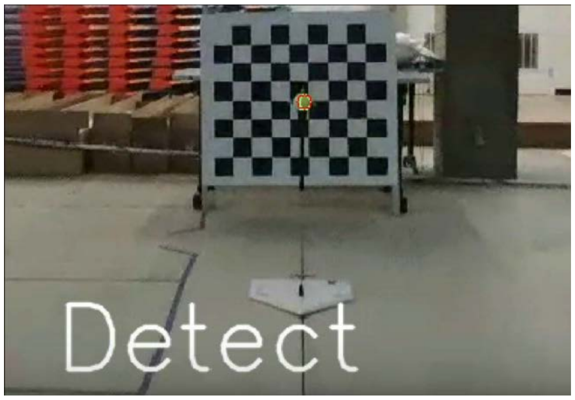


Figure 55. Detection frame with the average detection value and blob size being met

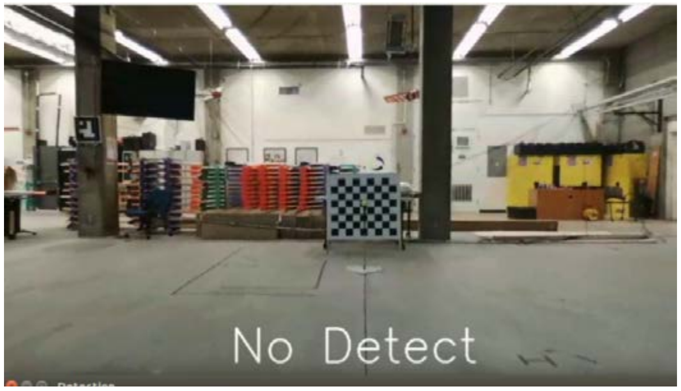


Figure 56. Full video background during the “robustness” test

Another interesting result that was seen during the “robustness” test was the video detectors ability to find the tennis ball prior to the operator’s eye being able to distinguish the tennis ball from the checker board background. This effect is illustrated in Figures 56–58.

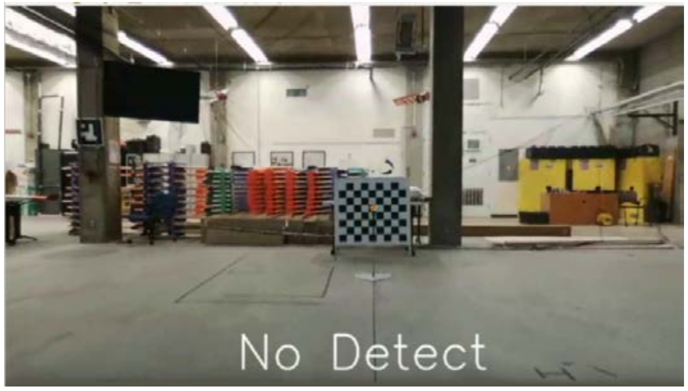


Figure 57. Initial detection in front of the entire background

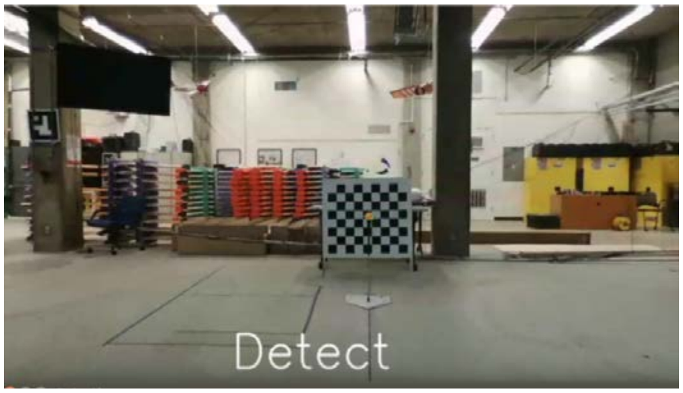


Figure 58. Full detection in front of the entire background

All these tests verified that the detection algorithm was robust to the environment and would not produce spurious detection during the full search and detection experiment.

C. KEY RESULTS OF PATH FOLLOWING AND SEARCH INTEGRATION

This section details the tests performed using the "mission controller." The "mission controller" is the culmination of the full motion controller and the video detection algorithm. The "mission controller" is the highest level of control and encompasses all the previously mention components. The first test was the "revisited pentagon" test which was the "pentagon" test with the target in the field of view of the camera. The next test was a generic boustrophedon search that has 5 lines and utilizes a 1-m sweep width. Next, is the "box" search which is a boustrophedon search with the sweep width of 3 m determined from the LRC Section E of Chapter IV. The last test is the boustrophedon search with the modified sweep width that considers the hidden areas that resulted from the flight geometry. Each of these tests correspond to the sweep width used to perform the complete coverage calculation. A wider sweep width resulted in a wider lawn mower pattern and conversely a narrower sweep width resulted in a tighter lawn mower pattern.

All these tests proved that the quadcopter could autonomously execute precise path following while using the video detector to identify targets within the search area. The defining characteristic of each of these tests is the sweep width that is used in order to calculate the search path pattern for each of the test.

1. Pentagon Revisited

The “revisited pentagon” test proved that the motion controller and the video detection algorithm were working together with known results from the video detector and the flight path. The results of the combined “revisited pentagon” test was expected to be just as precise as the “robustness” test and the “pentagon” test. In the “revisited pentagon” test, the target was placed within the detection view of the quadcopter and the pentagon flight pattern was flown again. The results of the flight test and the target location can be seen in Figure 59. The regions along the flight path where detections occurred are color-coded in green in Figure 59. The “revisited pentagon” test also produces tighter turns and transitions compared to the “pentagon” test. The improved performance of the quadcopter is due to the improved transit speed of the path follower and improved PID controller performance in the outer loop controller. These results can be seen in Figures 48 and 59.

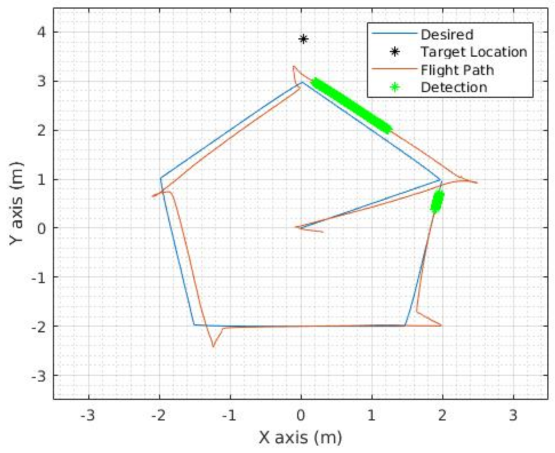


Figure 59. “Revisited pentagon” test with detection

During this test, the target was detected two times. The overall time history of detection for the “pentagon” test can be seen in Figure 60. The first detection occurred as the quadcopter was flying to the top point of the pentagon. The first detection can be seen in Figure 59 and 61. The second detection was right at the end of the pentagon shape when the target was at the left edge of the screen. The second detection can be seen in Figure 59 and 62. All results match or improve upon the results seen in the individual “robustness” and “pentagon” tests.

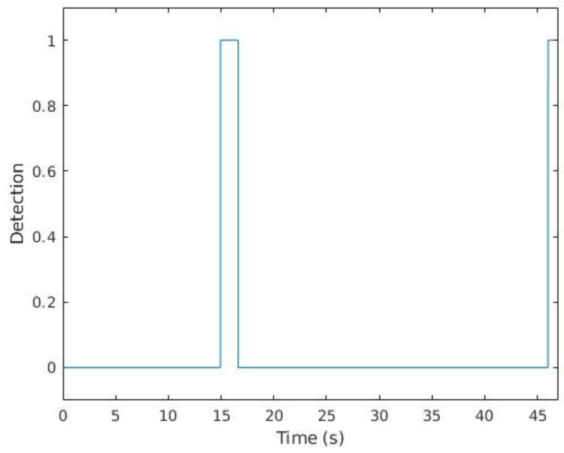


Figure 60. Time history plot of detection

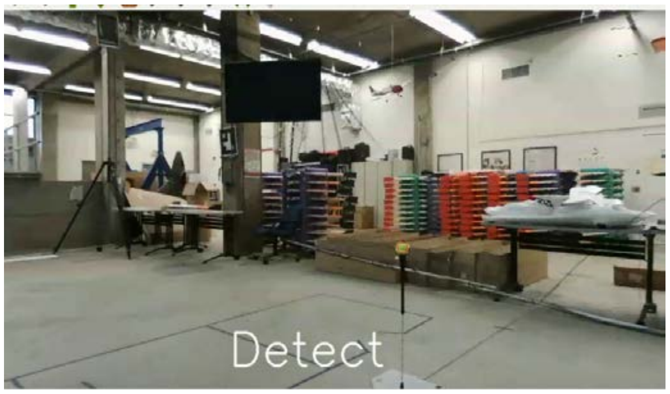


Figure 61. First detection during the “pentagon” test

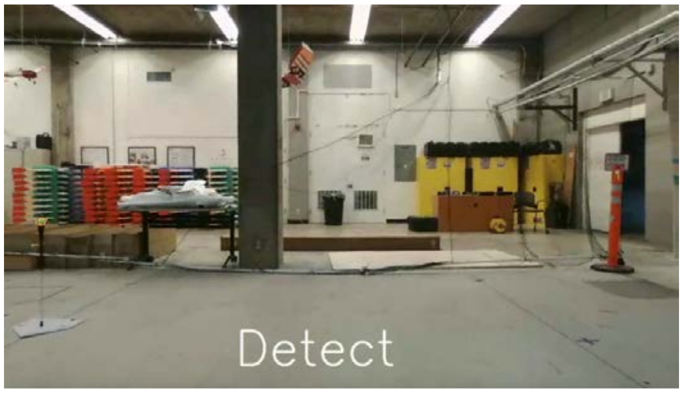


Figure 62. Second detection during the “pentagon” test

The “pentagon” test proved that all the components of the “mission controller” could work together and the target could be detected within the flight area. After the “pentagon” test was successful, the next test was the 5-Line Boustrophedon search.

2. 5-Line Search Pattern—The Boustrophedon Experiment

The “5-line search” test was the first test to take sweep width into consideration to provide complete coverage of the search area. The “5-line search” was performed with a sweep width of 1 m to over search the search area and to provide a base line test in which other experiments could be compared. For the “5-line search” test, the area being searched was a 2.5m by 2.5m box. The quadcopter started and finished at the origin of the search area and utilized the waypoints from the Path Planner. The flight path of the quadcopter can be seen in Figure 63.

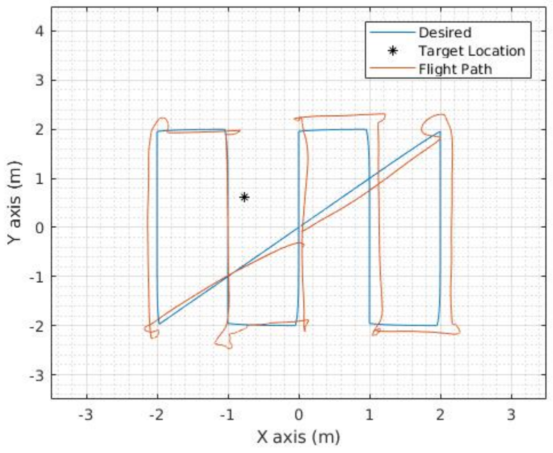


Figure 63. “5-line search” desired and actual flight path with target location

As expected, due to the over searching of the area, the target was detected along four different legs of the search. The detection results can be seen in Figure 64 where the detection regions are color-coded in green and overlaid on the flight path. The target location can also be seen in Figure 64 compared to the flight path. The time history plot of detection for the “5-line search” test can be seen in Figure 65. These results supported the LRC testing seen in Figure 45 in Chapter IV.

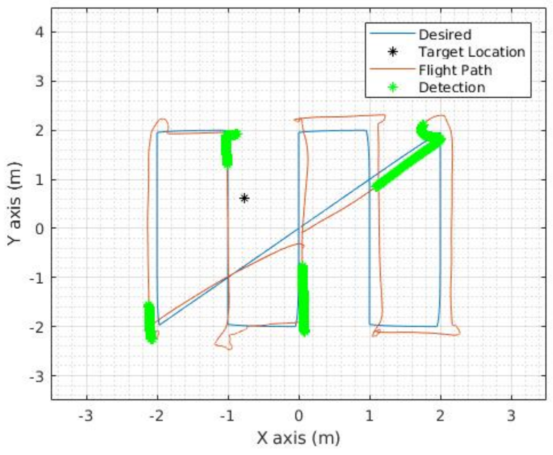


Figure 64. “5-line search” test with color coded detection and target location

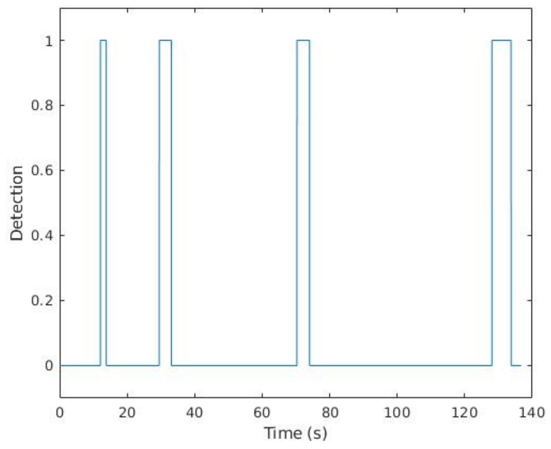


Figure 65. Time history plot of detection for the “5-line search” test

During the “5 line search” test the video detector had no spurious detections. The detections during this test occurred along different lines of the search or while turning onto the new path. Each detection had a different background and lighting conditions in each case. This flight path also confirmed the results seen in the “robustness” test of the video detector. The four detections for the “5-line search” test can be seen in Figures 66–69.

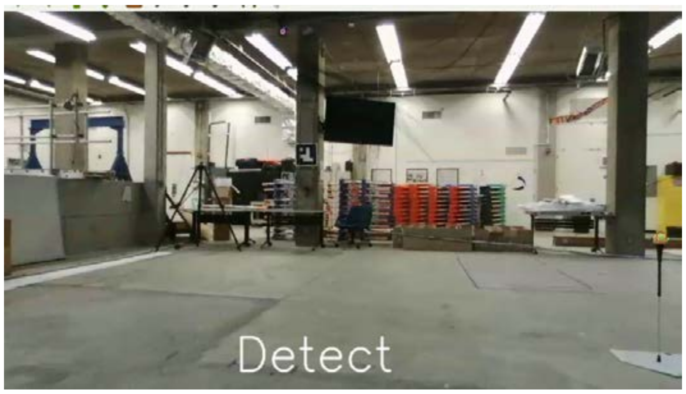


Figure 66. “5-line search” test first detection

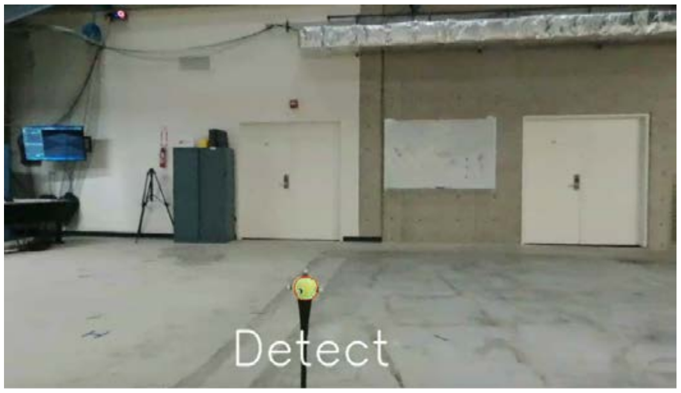


Figure 67. “5-line search” test second detection

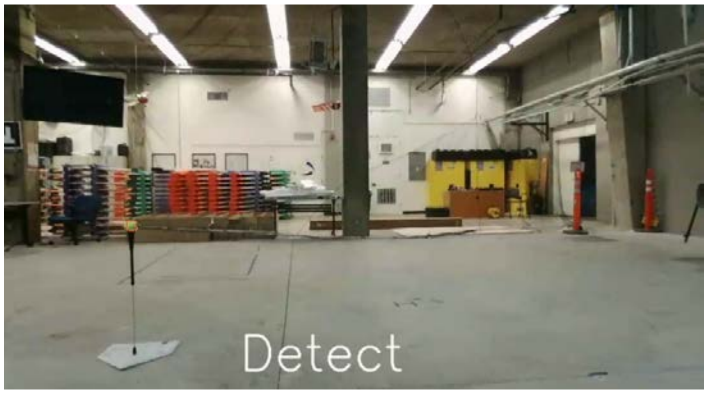


Figure 68. “5-line search” test third detection

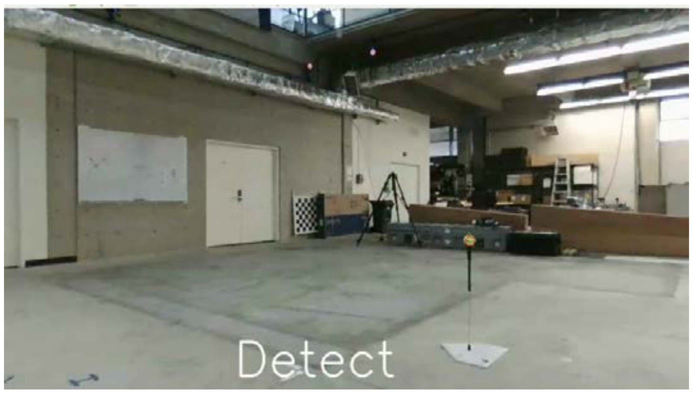


Figure 69. “5-line search” test fourth detection

The “5-line search” provided the base line in which the other lawn mower pattern searches will be compared.

3. Box Search Pattern

The "box search" test was the next evolution of the search pattern tests. The "box search" test was an attempt to maximize the initial sweep width of 3 m taken from the LRC data in Figure 45. The search area was 3 x 3 m. Due to the 3-m sweep width, the search path only required a 1.5 x 1.5 m box to be traced out one time. This test was run multiple times (more that 10 times) with repeatable success and it minimized the multiple detections seen in the "5-line search" test. This test also reduced the amount of time required to search a 3 x 3 m box. The overall flight time for the "box search" test was 46 seconds compared to 138 seconds in the "5-line search" test. The flight path for one of these tests can be seen in Figure 70.

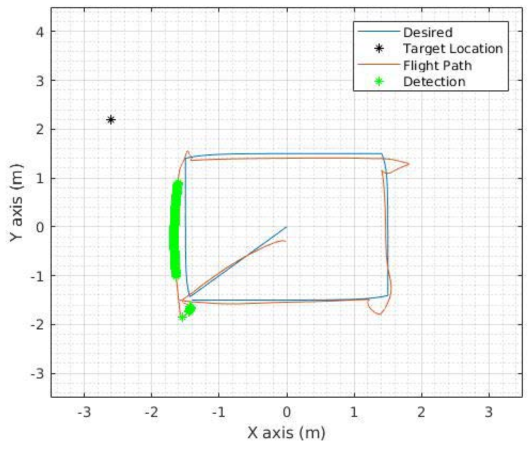


Figure 70. "Box search" test with detection

The target location and detection region for the test was recorded and can be seen in Figure 70. The time history plot of detection can be seen in Figure 71. In this "box search" test, the detection algorithm identified the target three times. All three of these detections occurred on the same leg of the search. The "box search" test gave a minimum search time for the 3x3 m area and was used to compare the results of the final experiment.

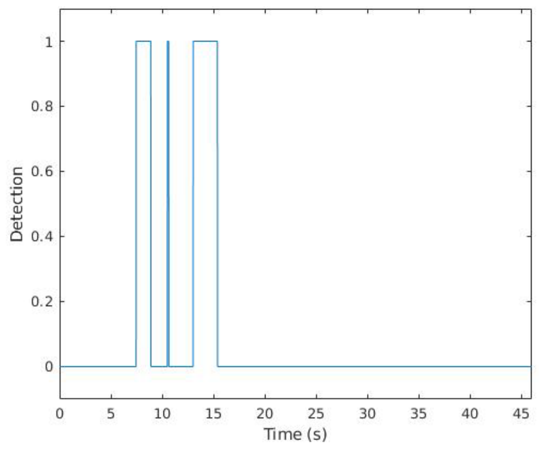


Figure 71. Time history plot of detection of “box search” test

4. Modified Sweep Width

During the analysis of the “box search” test it was determined that the 3-m LRC is only valid if the quadcopter is farther than 2.598 m from the target. This limitation is due to the triangular shape of the video detector in front of the camera. The video detector has a triangular shape. As the quadcopter turns, there may be a hidden region that is outside the search areas of two consecutive legs. The geometry of such a turn can be seen in Figure 72.

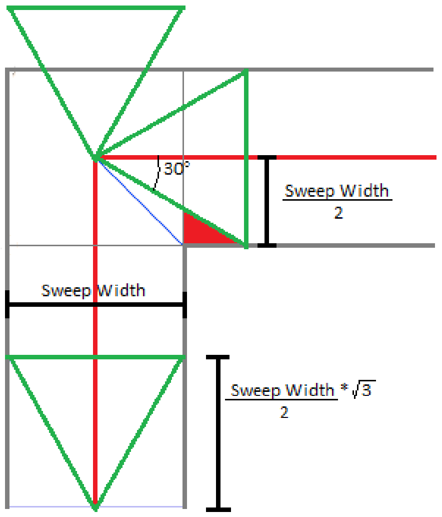

30°
$\frac{\text{Sweep Width}}{2}$
Sweep Width
$\frac{\text{Sweep Width}}{2} * \sqrt{3}$

Figure 72. Detector geometry from the "box search" experiment with red area representing a hidden spot.

In Figure 72, the green triangle is the swept area of the detection algorithm used to cover the search area. The quadcopter has a field of view of 60 degrees total or 30 degrees on either side of the centerline. The red line in Figure 72 is the flight path of the quadcopter. When the quadcopter gets to the end of one segment, the green swept area rotates about the turning point of the quadcopter. The initial and final geometry of the detector through the turn can be seen in Figure 72 by the two green triangles. In Figure 72, the solid red triangle is the missed search area. This missed search area is a result of the 30 degree half field of view. If the camera had a 90-degree field of view this would not result in a missed search area.

The base of the red triangle can be determined in generic terms of sweep width utilizing geometry. The hidden triangle geometry can be seen in Figure 73. In

$$\tan(30^\circ) = \frac{\frac{sweepwidth}{2}}{\frac{sweepwidth}{2} + trianglebase} \tag{19}$$

the half sweep width acts as the opposite side of the triangle and the adjacent side of the triangle is composed of the half sweep width and the base of the red triangle. Using (19),

$$triangle base = \frac{\frac{sweepwidth}{2}}{\tan(30°)} - \frac{sweepwidth}{2} \tag{20}$$

can be expressed in terms of sweep width. Rearranging (20), the

$$triangle base + \frac{sweepwidth}{2} \leq 1.5 meters \tag{21}$$

must be true if the maximum sweep width is 1.5 m. After substituting (20) into (21)

$$\frac{\frac{sweepwidth}{2}}{\tan(30°)} - \frac{sweepwidth}{2} + \frac{sweepwidth}{2} \leq 1.5 meters \tag{22}$$

is presented in terms of the new sweep width. Solving (22) results in a new full sweep width that is less than 1.73 m. Any search paths that use the hidden triangle geometry and use a 3 m sweep width would cause a missed detection.

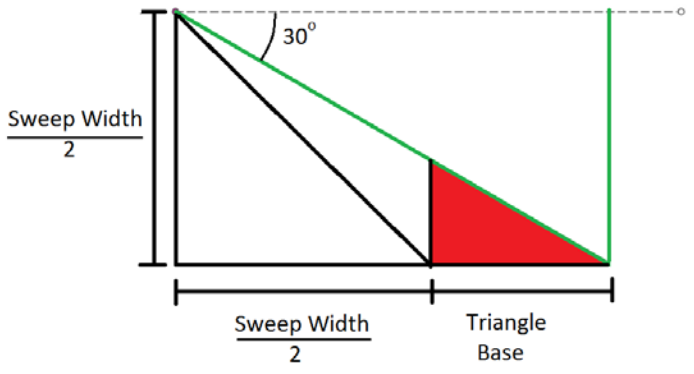


Figure 73. Hidden triangle during search

5. Hidden Triangle Search

The “hidden triangle” test was designed to exploit the hidden triangle mentioned in the previous section. During the “hidden triangle” test, the target was not detected on any of the legs of the test. As the quadcopter made a turn with the target inside of the hidden triangle, the target only just broke the edge of the video frame. The quadcopter has a 30-degree half angle view on either side of the centerline. The target was initially detected at

28.62 degrees but was not in the video frame long enough to meet the average frames requirement. The flight path for this test can be seen in Figure 74. The time history plot of detection can be seen in Figure 75 which shows no detection for this test.

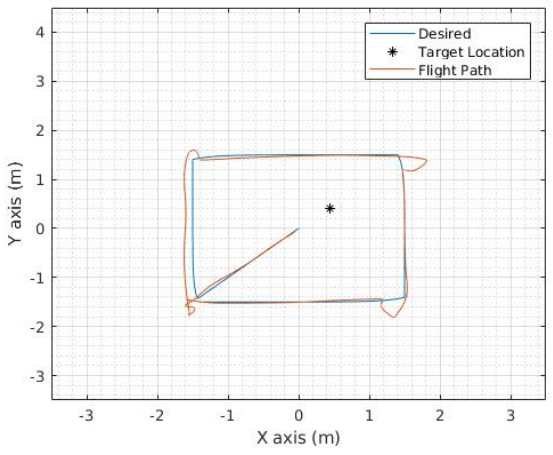


Figure 74. “Hidden triangle” test with detection

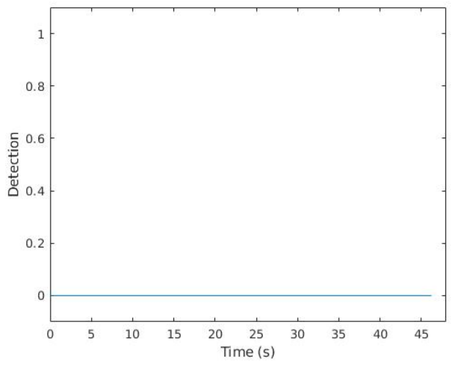


Figure 75. Time history plot of detection for the “hidden triangle” test

A full detection was never made but the video detection algorithm did see the object. The target was not within the video frame long enough to meet the 7 out of 14 frame requirement for full detection. This initial detection of the target can be seen in Figure 76. The “hidden triangle” test demonstrated hidden triangle does exist when the quadcopters make a turn. This test also proves the robustness of the detection algorithm as it also did not produce any spurious detection.

Figure 76. “Hidden triangle” test initial detection

The results from the “hidden triangle” test and the calculation from section 4 confirmed that a hidden triangle did exist at each corner of the box search pattern. In order to eliminate the hidden triangle, the boustrophedon search was recalculated. The new half sweep width used for this search path was 1.7 m. The new sweep width is 0.03 m less than the maximum half sweep width determined in (22). The results of the search pattern can be seen in Figure 77. This search path has 4 lines in the y-direction and will be referred to as “4-line search” from this point forward. This new search algorithm offers complete coverage during a search with sharp turns.

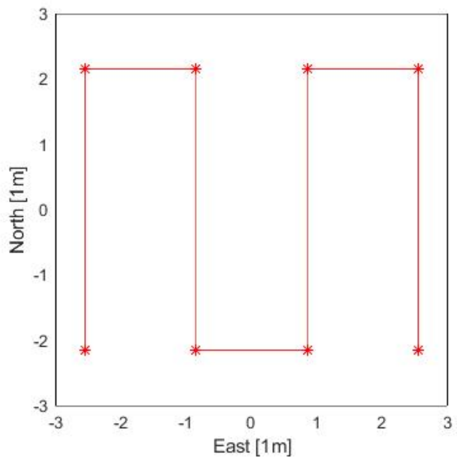


Figure 77. “4-line search” path

6. 4-Line Search Test

The “4-line search” utilizes a sweep width of 1.7 m to calculate the boustrophedon pattern for the search path. The “4-line search” test provides complete coverage of a 3 x 3 m search area. The “4-Line search” is not as efficient at finding the target as the “box search” test, but it eliminates the possible miss of targets within the hidden triangle. The “4-line search” is also more efficient than the “5-line search” in terms of detections and time of flight. The “4-line search” test results can be seen in Figure 78.

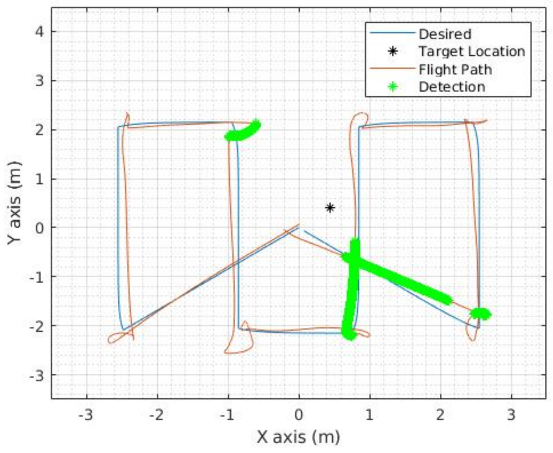


Figure 78. “4-line search” test results with detection and target location

The “4-line search” test took 116 seconds compared to the 46 seconds in the “box search” test and the 138 seconds in the “5-line search” test. Figure 79 shows the target being detected four times over three different legs of the “4-line search” test. Each detection for the “4-line search” test can be seen in Figures 80–82.

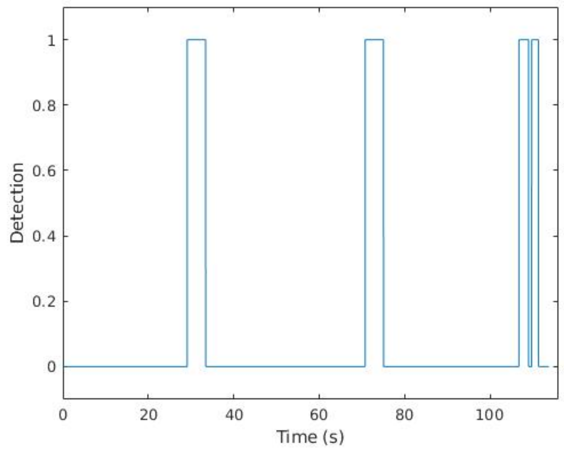


Figure 79. Time history plot of detection for the “4-line search” test

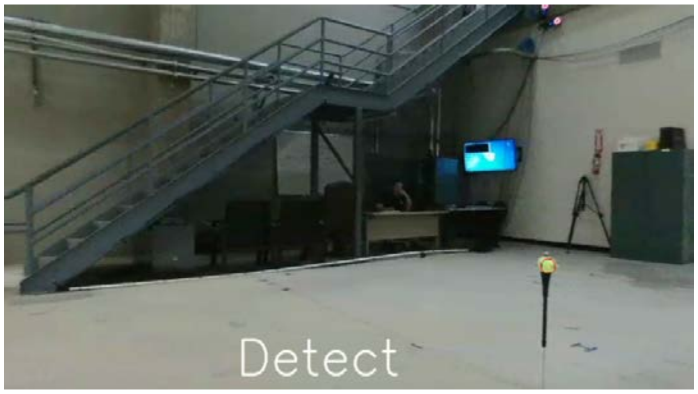


Figure 80. “4-line search” test first detection

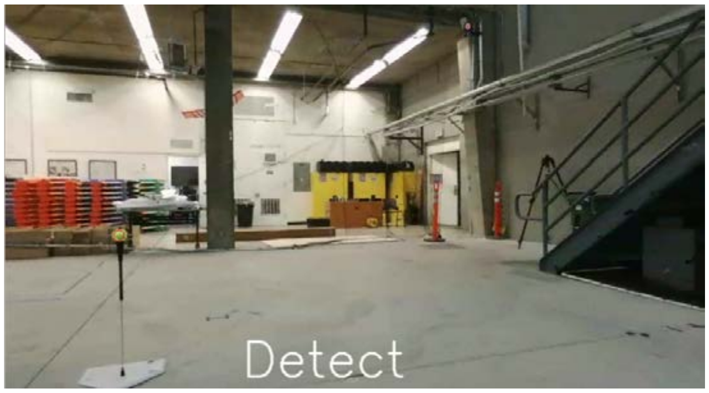


Figure 81. “4-line search” test second detection

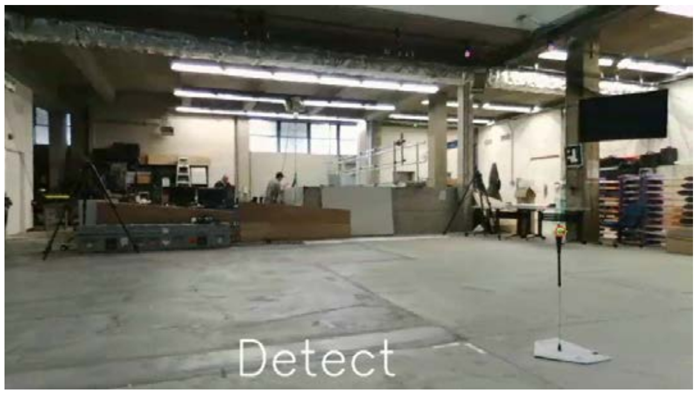


Figure 82. “4-Line search” test third detection

D. CONCLUSION OF EXPERIMENTAL SECTION

1. Controller Results

Overall, the flight dynamics of the quadcopter greatly affect the performance of the video detection algorithm. If the quadcopter flies too fast, then the target will be missed. If it flies to slow, then the flight time of the mission will be extended. If the quadcopter cannot maintain its position along the path, then the target may be missed. Therefore, a robust and precise flight controller is needed in order to perform autonomous passive video sensing.

The current “mission controller” provides stable flight control and allows for accurate detection of targets within a search area. The quadcopter can be controlled within an average of 20cms from the desired flight path. The target can be detected with a 1.7 m sweep width and up to a 3 m sweep width while on long straight flights. The triangular detection region can be used to optimize the search path to provide complete coverage and minimize mission time. The detection algorithm was able to identify the target and quantify detections with a variety of backgrounds. None of the experiments produced a spurious detection. Each of the experiments contributed or verified the building blocks to assemble the final “mission controller.”

2. Hardware and Software Considerations

The Parrot Bebop quadcopter provides a rugged and robust platform with higher level capabilities that can be used in an autonomous capability. The Parrot Bebop is designed for taking video with slower more gradual flight dynamics. The overall flight dynamics were set to the maximum possible values allowed by the Parrot software to provide the fast flight dynamics. Despite the numerous crashes throughout testing, not a single component was damaged to the point of needing to be replace. The durability of the Bebop is attributed to Parrot Inc’s overall design and quality.

The Bebop’s communication structure allows for adequate data transmission in a timely manner to provide the feedback needed for autonomous flight. The Simulink coding environment allowed for rapid prototyping and numerous restructurings that ended with the final “mission controller.” The coding flexibility of Simulink made the frequent design and implementation of code quick and efficient. The Simulink code is processor and RAM intensive and the “mission controller” would have to be moved to another programming language to be run on a less capable computing platform.

ROS provided the communication paths to implement multiple programming languages, running on two different platforms, to complete the overall control structure that allowed for autonomous flight. The high-quality video sensor onboard the Bebop is more than adequate to be used as a passive video sensor and to aid in the control and mission of the autonomous quadcopter.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. EXPERIMENTAL CONCLUSION

The experiment started with a COTSQ that was meant to be flown by an operator with a transmitter and had no autonomous capability. A software and hardware infrastructure was developed around the Bebop to enhance its existing capabilities. With this infrastructure, the COTSQ was converted to an autonomous search platform with the capability of generating a search trajectory according to the onboard detection capabilities and its flight dynamics. The communication infrastructure of the research setup integrated ROS, Simulink, Python, and the quadcopter. For a given search area, the “mission controller” takes in the area geometry and the passive sensor characteristics to calculate the desired search pattern. The “mission controller” then provides the controls to fly the quadcopter through that search area autonomously. The motion controller also has the built-in capability to enable the same autonomous search mission in an outdoor environment.

The video detection algorithm has been proven robust for the task of search of single and multiple targets; the algorithm is parameterized to enable different number and general descriptive characteristics of possible target. The video detector can detect a single/multiple target with the vision-based framework. The LRC calibration of the video sensor has been tested and verified with numerous experimental results of the sweep width being accurate and robust. Overall, the COTSQ was able to be converted to autonomous flight and perform a military type mission of searching and detecting a target. With further testing and research, COTSQs can be a viable alternative for the larger and more expensive autonomous platforms that the military uses today.

B. EFFECTIVENESS OF SEARCH

After performing abundant searches with the Bebop quadcopter, the COTSQ was able to successfully detect the target in every mission that the sensor could see the target and had time to detect the target. The video detection algorithm was proven robust to spurious detections and during the LRC testing it showed perfect detection results within

3 m. The repeatability of the flight dynamics was also shown during the LRC test with less than 15 cm error along the flight path and 10 cm error across the flight path between each of the passes. The design of the search algorithm and the creation of the flight path can still be optimized but overall the "mission controller" is very reliable and effective at finding the target along the flight path.

C. MOTION CONTROLLER IN FUTURE WORK

1. Modifying Control Architecture of the Autopilot

The flight dynamics is one of the most critical portions of the search algorithm when it comes to passive video detection. Any flight that is outside of the optimal path has a penalty in efficiency, flight time, and possible introduction of spurious or missed detections. The current motion controller has predictable flight dynamics and search capability. One way that this motion controller could be improved for future work is by optimizing the autopilot architecture and its tuning in all control channels. The path optimization would increase the efficiency of the search and detection procedure.

Specifically, the control commands interface provided by Bebop does not provide the desired capability of accepting the low-level reference commands for the autopilot. In the current state, the accepted commands include values proportional to the desired motion along 3 axes of the body reference frame and the total speed command. The "analytical conversion" of those commands into internal autopilot signals (like reference rates, speed, and accelerations) are not exposed to the end user. While this approach certainly makes the UAV safer and user-friendly in recreational flying, the approach does not allow developing more aggressive and higher capability control algorithms.

2. Outdoor Search

The "motion controller" and the Parrot Bebop quadcopter can switch to outdoor flight using the VIO and GPS data as discussed in Chapter III. Testing the outdoor flight capabilities of the Parrot Bebop to perform a similar search outdoors would further prove the robustness of the motion controller and the detection algorithm. This would also be a logical next step in getting COTSQ into the hands of military operators. GPS data has 2 orders of magnitude lower accuracy than the VMCS. The reduced accuracy means the "motion controller" would have to be augmented the positional feedback with an Extended Kalman Filter to provide a similar level of positional feedback.

D. VIDEO DETECTOR IN FUTURE WORK

1. Decreased LRC Size

In all the searches, the target size produced a large sweep width compared to the search area and thus produced a relatively quick search pattern. The large sweep width and small search area aided in the rapid prototyping of the motion controller. Now that the "motion controller" and the video detector have been fully developed, a decrease in the sweep width would provide a more thorough examination of search path patterns and optimization. Another way to accomplish this same result would be to increase the operational search area. Either decrease sweep width or increased search area would also increase the desire for cross-discipline collaboration between the operations research (OR) and the mechanical engineering (ME) fields.

2. Use Video Detection Cone to Optimize the Search Path

The "cookie cutter" sensor that was used in this thesis provided an adequate ability to define the sweep width and to allow a path to be designed using this sweep width. A better approach would be to use the actual pyramid shape of the video detector. Using this specific shape to optimize the search path would increase the efficiency of the actual quadcopter flight.

3. Use the Pan and Tilt Function to Cover the Hidden Triangle

The HD camera on the front of the Bebop has a digital capability to pan and tilt the image. This capability was not analyzed or tested during this thesis. One possible use of this functionality would be to use the pan function to cover the hidden triangle during the turns of the quadcopter without commanding the UAV to perform a full 360-degree rotation in yaw. The hidden triangle is only a result of the narrow field of view compared to the turn of the quadcopter. By using the pan of the video frame during turns in the same direction as the turn, the hidden triangle could be eliminated. Covering the hidden triangle digitally would allow the restriction on the LRC to be relaxed which would also greatly increase the efficiency of the search.

4. Output the Target Location, Percent of Detection, and Timestamp So a User Can Later Go Back and View the Video of the Event

The goal of this thesis is to build the tools so that a user does not have to sit in front of a video feed and watch it in real time. The video feed that is output with the flag of detect/no detect is not meant for the user but is a tool that is used to iterate the capability of the detection algorithm. Ultimately, the video feed would not be displayed in real time but would but recorded with the overlays and masks. The time history plot of detection would then be used to earmark the video timeline so that an operator could go directly to the key events and review it in a fraction of the time. Additional to this earmark on the video, the centroid and radius of the target along with the GPS data of the quadcopter could provide a location of the target. All this information could be coupled so that the user that wants to view this afterward can access all this information at one time.

5. Machine Learning and Object Recognition

The capability for machine learning and object recognition has greatly increased with parallel computing and algorithm approaches in the last 20 years. For example, OpenCV has already been used to accomplish these tasks on station and dynamic platforms. Another logical step would be to expand the capability of the detection algorithm to start recognizing objects through machine learning. The video detection algorithm could be trained on thousands of pictures of IEDs or explosives and when the quadcopter flies through the area it could provide a confidence of detecting an explosive.

The machine learning and object detection algorithms could also be used during search and rescue for survivors of a natural disaster. The detection algorithm could be trained to look for humans instead of targets. Matlab has an example of face recognition that uses machine learning [19]. Again, the detection algorithm could provide a confidence of detection to an operator.

6. Indoor SLAM Using the Video Detector

The COTSQ quadcopters naturally lend themselves to indoor searches due to their size and maneuverability. The video feed could not only be used to detect a single target but could be used to detect all obstacles. A map of the obstacles could then be created. This map could then be used by additional autonomous platforms or operators to go into the building afterwards.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. COMMUNICATION SETUP

A. BEBOP_AUTONOMY PACKAGE INSTRUCTIONS

1. Go to https://bebop-autonomy.readthedocs.io/en/latest/installation.html

2. Follow directions for installation as the commands or git repository may have changed.

3. Enter the following commands in the terminal:

   a. sudo apt-get install build-essential python-rosdep python-catkin-tools
   b. mkdir -p ~/bebop_ws/src && cd ~/bebop_ws
   c. catkin init
   d. git clone https://github.com/AutonomyLab/bebop_autonomy.git/src/bebop_autonomy
   e. rosdep update
   f. rosdep install --from-paths src -i
   g. catkin build

4. Once the Bebop_Autonomy Software Package has been installed. You will need to setup the quadcopters for single or multiple quadcopter use.

B. SINGLE QUADCOPTERS

The bebop acts as the host for the UDP communications between the quadcopter and the ground station. If you are only flying one quadcopter then you can use the stock IP address of 192.168.42.1 and you will not need to update the launch files.

C. MULTIPLE QUADCOPTERS

If you are going to be flying more than one quadcopter at the same time you will need to change the stock IP address on one or both quadcopters. You will also need to change the IP address in the launch files in the /bebop_ws/src/bebop_autonomy/ bebop_driver/launch folder to match the IP address that you put on the quadcopter.

Background: For the ground station to connect and communicate with both quadcopters, they need to be on individual subnets. The first three sets up numbers must be individual from one another for example 192.168.101.1 and 192.168.102.1.

In this experiment is used the following IP addresses for the three quadcopters.

A=192.168.101.1
B=192.168.102.1
C=192.168.103.1

This allows for individual subnets so that each bebop can connect to the laptop individually

D. CHANGING THE IP ADDRESS FOR MULTIPLE QUADCOPTERS

Reference https://github.com/tnaegeli/multiple_bebops to
change the IP address

Instructions to change the IP address on each quadcopter:

1. switch the drone on and connect your pc to the drone access point
2. open a terminal in Linux
3. Connect the drone to the computer via a USB cable
4. press the on switch 4 times rapidly
    a. make sure that adb is installed
        i. if not use
            1. sudo apt install adb
5. type "adb connect 192.168.43.1:9050" into the terminal window
6. type "adb shell"
7. type "mount -o remount,rw /"
8. type " cd sbin" (there is a space in front of cd sbin)

9. type “vi broadcom_setup.sh”

    i. move the cursor to the IP address and type i to start writing, esc to stop and wq to exit.

10. change field IFACE IP AP=“192.168.42.1” to IFACE IP AP=“192.168.xxx.x,” where x represents any number which you have not assigned to other drones

11. hit esc to exit the writing interface

12. type “:wq” hit enter

13. save the file and exit

E. CHANGING BEBOP LAUNCH FILES FOR MULTIPLE QUADCOPTERS

1. Navigate to the folder /bebop_ws/src/bebop_autonomy/bebop_driver/launch

2. Open bebop_node.launch file for editing

3. Change the line
   <arg name=“namespace” default=“bebop”/>
   to
   <arg name=“namespace” default=“bebopA”/>
   where A is a new number or letter to designate this bebop from the rest

4. Change the line
   <arg name=“ip” default= “192.168.42.1”/>
   to line
   <arg name=“ip” default= “192.168.xxx.x”/>
   where the x’s match the x’s that was assigned to the bebop.

5. Save this as a new file with the name bebop_node_A.launch where A is the same number or variable that you used above.

6. This is the new launch file that you will use to communicate with this bebop

7. Each Bebop will have to have its own launch file running before communication will be established

F. SOURCING THE BEBOP FOR ROS

This will change the bashrc file so that the bebop_ws does not need to be sourced everytime you open the terminal window.

1. Open the .bashrc file by typing “gedit .bashrc”

2. Place the following line at the end of the bashrc file
    i. source /home/cavr/bebop_ws/devel/setup.bash

3. Save the file

4. Close all terminal windows

5. Reopen a terminal and the bebop_ws should not need to be sourced after this point

APPENDIX B. ESSENTIAL COMMANDS FOR BEBOP

For a full understanding of the commands that Parrot Bebop receives via the bebop_autonomy package reference https://bebop-autonomy.readthedocs.io/en/latest/

A. CONNECTING TO THE BEBOP

1. Connect to bebop Wi-Fi
    a. This can be done via the computer Wi-Fi connection or a Wi-Fi dongle. The bebop will show up as a network in the network list. Each individual quadcopter requires its own Wi-Fi network. This was accomplished with multiple Wi-Fi dongles to a single ground station in this thesis.

2. Open a terminal window

3. Source the launch files (this must be done if the bashrc file was not changed)
    a. source ~/bebop_ws/devel/setup.bash

4. Then run a launch file for each bebop (change the following to match the file name of the launch file)
    a. roslaunch bebop_driver bebop_node.launch
    b. roslaunch bebop_driver bebop_node_A.launch
    c. roslaunch bebop_driver bebop_node_B.launch
    d. roslaunch bebop_driver bebop_node_C.launch

B. BASIC COMMAND LINE COMMUNICATION WITH THE BEBOP

1. Takeoff
    a. rostopic pub --once /bebop/takeoff std_msgs/Empty
    b. rostopic pub --once /bebopA/takeoff std_msgs/Empty
    c. rostopic pub --once /bebopB/takeoff std_msgs/Empty
    d. rostopic pub --once /bebopC/takeoff std_msgs/Empty

2. Land
    a. rostopic pub --once /bebop/land std_msgs/Empty
    b. rostopic pub --once /bebopA/land std_msgs/Empty
    c. rostopic pub --once /bebopB/land std_msgs/Empty
    d. rostopic pub --once /bebopC/land std_msgs/Empty

3. Emergency Stop
    a. rostopic pub --once /bebop/reset std_msgs/Empty
    b. rostopic pub --once /bebopA/reset std_msgs/Empty

   c. rostopic pub --once /bebopB/reset std_msgs/Empty
   d. rostopic pub --once /bebopC/reset std_msgs/Empty

Reference https://bebop-autonomy.readthedocs.io/en/latest/ for an understanding of the values that can go into the next communications.

4. Zero Linear Velocity and Angular Velocity command
   a. rostopic pub -1 /bebopA/cmd_vel geometry_msgs/Twist –'[0.0, 0.0, 0.0]' '[0.0, 0.0, 0.0]'

5. Zero Linear Velocity and Angular Velocity command
   a. rostopic pub -1 /bebopA/cmd_vel geometry_msgs/Twist –'[linear.x, linear.y, linear.z]' '[angular.z, 0.0, 0.0]'

6. Camera Command
   a. rostopic pub -1 /bebopA/camera_control geometry_msgs/Twist -- '[0, 0, 0]' '[0,0,0]'

7. Flip
   a. rostopic pub -1 /bebopA/flip std_msgs/UInt8 – 0
   b. rostopic pub -1 /bebopA/flip std_msgs/UInt8 – 1
   c. rostopic pub -1 /bebopA/flip std_msgs/UInt8 – 2
   d. rostopic pub -1 /bebopA/flip std_msgs/UInt8 -- 3

C. CHANGE THE DYNAMIC CONTROLS TO MAKE BEBOP MORE ACROBATIC

This change must be executed every time that you run the launch file.

1. rosrun rqt_reconfigure

2. Setting values
   a. pilotingSettingsMaxTiltCurrent= 90
   b. PilotingSettingsAbsolutControlOn=1 on
   c. SpeedSettingsMaxVerticalSpeedCurrent= 10
   d. SpeedSettingsMaxRotationSpeedCurrent= 900
   e. SpeedSettingsMaxPitchRollRotationSpeedCurrent=900

3. Click on the X in upper right corner to save changes

D. VIEW THE VIDEO FROM THE BEBOP VIA ROS FROM COMMANDLINE

1. rosrun image_view image:=/bebopA/image_raw

APPENDIX C. VIDEO DETECTOR

This appendix goes over aspects of the video detector.

A. SIMULINK CODE FOR ROS VIDEO PROCESSING

This method was too cumbersome to be run in conjunction with the flight controller, but might be useful for less processing intense controllers. The full Simulink algorithm for video detection can be seen in Figure 83.

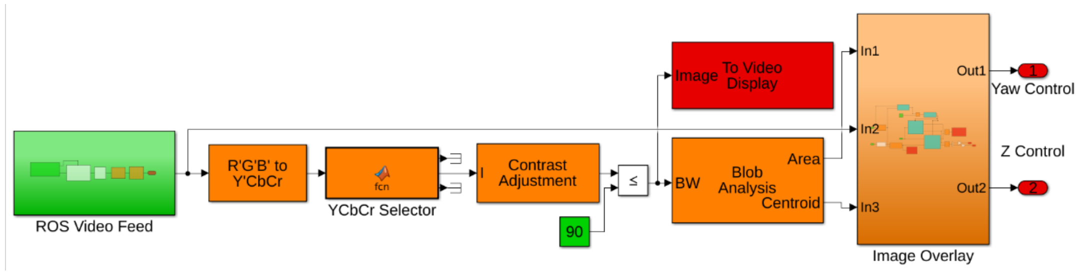


Figure 83. Full Simulink video detector

The most important piece of this code is the ROS Video Feed. This block subscribes to a ROS topic with video data and then performs the necessary matrix conversion to change the data from a string array to an RGB image matrix. The image that was used in this thesis is an 856x480x3 RGB image. It is transmitted via the ROS topic as a 1x1232640 linear array. (Note in order to get the full array you must change the maximum size of arrays in Simulink settings). The array size will be different if you have a different size RBG image. The reshape block segments this linear array into 3 matrices. These matrices are not in the correct order and must be reordered to produce the correct RGB image. The full algorithm can be seen in Figure 84.

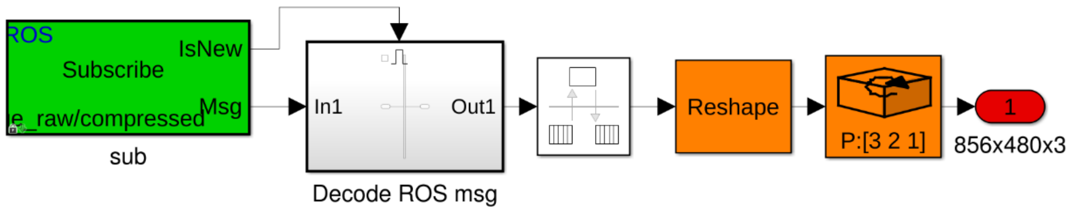


Figure 84. ROS topic subscription and matrix transformation

The logic that applies the Detect or No Detect text is a simple if statement that tests for a blob of the correct size. The Detect or No Detect text is then merged with the original video and output to a display or sent to perform some control function. The logic can be seen in Figure 85.

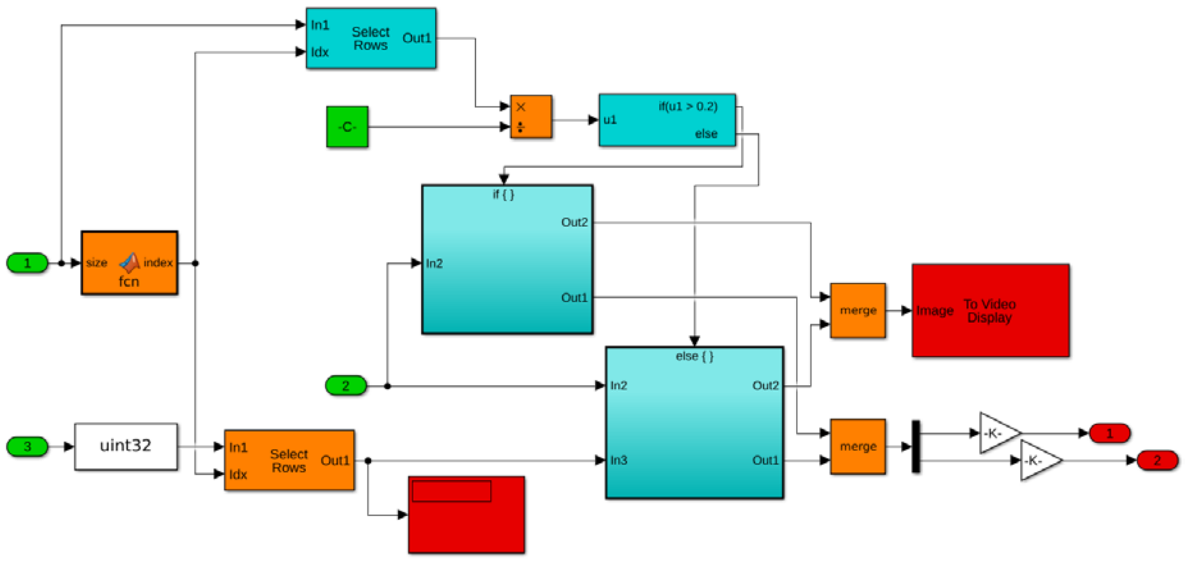


Figure 85. Simulink logic to apply detect/no detect to video

B. FINDING COLOR CODE FOR IMAGE MASK

This section goes over the process of using the colors in the video feed from ROS to determine the HSV upper and lower mask values

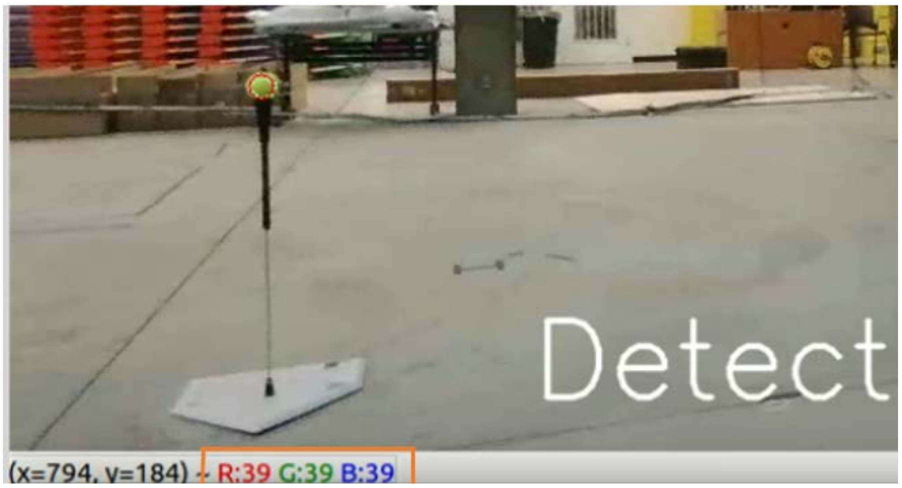
Connect the ground station to the Bebop via Wi-Fi

Run “roslaunch bebop_driver bebop_node_A.launch” from the command line

Run “rosrun image_view image:=/bebopA/image_raw” from the command line

Mouse over the object that you are trying to detect. The RGB values change depending on where the mouse places the x and y values within the image

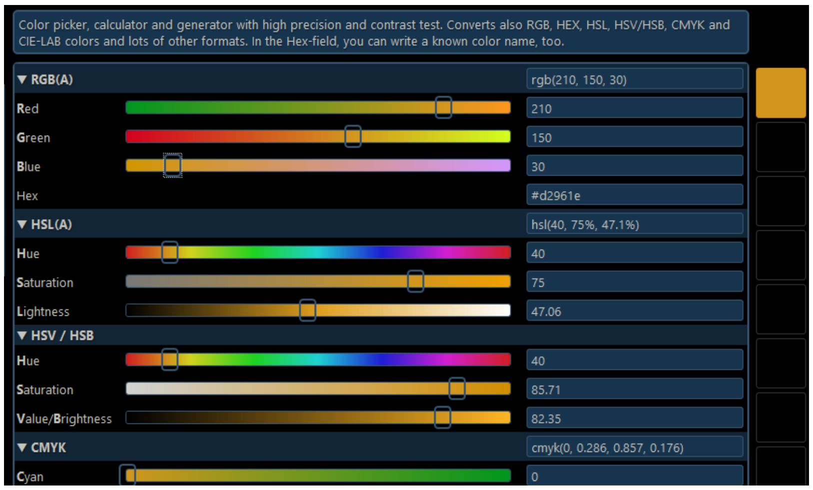
Record the RGB values seen in Figure 86



```
(x=794, y=184) ~ R:39 G:39 B:39
```

Figure 86. Video Feed with RGB values

Go to http://colorizer.org/ seen in Figure 87 [20]



Color picker, calculator and generator with high precision and contrast test. Converts also RGB, HEX, HSL, HSV/HSB, CMYK and CIE-LAB colors and lots of other formats. In the Hex-field, you can write a known color name, too.

▼ RGB(A)	rgb(210, 150, 30)
Red	210
Green	150
Blue	30
Hex	#d2961e
▼ HSL(A)	hsl(40, 75%, 47.1%)
Hue	40
Saturation	75
Lightness	47.06
▼ HSV / HSB	
Hue	40
Saturation	85.71
Value/Brightness	82.35
▼ CMYK	cmyk(0, 0.286, 0.857, 0.176)
Cyan	0

Figure 87. Colorizer.org color palette. Source: [20].

Change the RGB sliders to match the RGB values that were recorded from the video

Record the HSV value that corresponds to the RGB value. This is a starting place for a center value of the maximum and minimum values of the color mask

LIST OF REFERENCES

[1] Chief of Naval Operations, “A design for maintaining maritime superiority Version 2.0,” Dec. 2018. [Online]. Available: https://www.navy.mil/navydata/people/cno/Richardson/Resource/Design_2.0.pdf

[2] M. Goodrich. “Supporting wilderness search and rescue using a camera-equipped mini UAV.” *Journal of Field Robotics*, vol. 25, no. 1 pp. 89–110, 2008. [Online]. doi: 10.1002/ROB.20226

[3] S. Waharte, and N. Trigoni, “Supporting search and rescue operations with UAVs,” *2010 International Conference on Emerging Security Technologies*, Canterbury, 2010, pp. 142–147. doi: 10.1109/EST.2010.31

[4] K. Boudjit, and C. Larbes, “Detection and implementation autonomous target tracking with a Quadrotor AR.Drone,” *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO),* Colmar, 2015, pp. 223–230.

[5] E. Lopez, S. Garcia, R. Barea, L. M. Bergasa, E. J. Molinos, R. Arroyo, E. Romera, and S. Pardo, “A multi-sensorial simultaneous localization and mapping (SLAM) system for low-cost micro-aerial vehicles in GPS denied environments,” *Sensors(Basel),* vol. 17, no. 4, pp. 802. [Online]. doi: 10.3390/s17040802

[6] Parrot Inc, *Parrot Bebop Drone User Guide*, 2014. [Online]. Available: https://www.wellbots.com/content/Parrot/Parrot%20Bebop%20Drone%20User%20Guide.pdf

[7] Amazon, “Parrot Bebop Quadcopter Drone with Sky Controller Bundle (Blue).” Accessed Jan. 2019. [Online]. Available: https://www.amazon.com/Parrot-Bebop-Quadcopter-Controller-Bundle/dp/B00OOR911E

[8] ROS, “About ROS,” Accessed Jan. 2019. [Online]. Available: http://www.ros.org/about-ros/

[9] K. P. Valavanis, and G. J. Vachtsevanos, *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer Netherlands, 2015.

[10] N. L. M. Jeurgens, “Identification and Control Implementation of an AR. Drone 2.0,” M.S. thesis, Dynamics and Control, Dept of ME, Eindhoven UT, The Netherlands, 2017. [Online]. Available: https://dc.wtb.tue.nl/lefeber/do_download_pdf.php?id=176

[11] N. S. Nise, *Control Systems Engineering*. Hoboken, NJ, USA: Wiley, 2015.

[12] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB®*, 2nd ed. United States: Gatesmark Publishing, 2009.

[13] OpenCV, "About OpenCV" Accessed Jan 2019. [Online]. Available: https://opencv.org/about.html

[14] P. Corke. *Robotics, Vision and Control Fundamental Algorithms in MATLAB.* New York, NY, USA: Springer, 2011.

[15] MathWorks, "About Stateflow," Accessed Jan 2019. [Online]. Available: https://www.mathworks.com/products/stateflow.html

[16] A. R. Washburn. *Search and Detection*, 5th ed. Monterey, CA, USA: Operations Research Department, Naval Postgraduate School, Monterey, CA 2014.

[17] E. M. Arkin, S. P. Fekete, J. S. B. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry,* vol. 17, no. 1–2, pp. 25–50, Oct 2000. [Online]. Available: https://ac.els-cdn.com/S0925772100000158/1-s2.0-S0925772100000158-main.pdf?_tid=f6227e60-2681-4653-88e5-b020e83038f9&acdnat=1549992227_dc36eb8d8a7c0d34853be38cbf7bbdd5

[18] S. W. Moon, D. H. Shim, "Study on Path Planning Algorithms for Unmanned Agricultural Helicopters in Complex Environments,"*International Journal for Aeronautical and Space Sciences*, vol. 10, no. 2, pp. Nov 2009. [Online]. doi: 10.5139/IJASS.2009.10.2.001

[19] MathWorks,, "Face Detection and Tracking Using CAMShift," Accessed Feb 2019. [Online]. Available: https://www.mathworks.com/help/vision/examples/face-detection-and-tracking-using-camshift.html.

DoD] Colorizer.org, "Color Picker," Accessed Feb 2019. [Online]. Available: http://colorizer.org/

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California